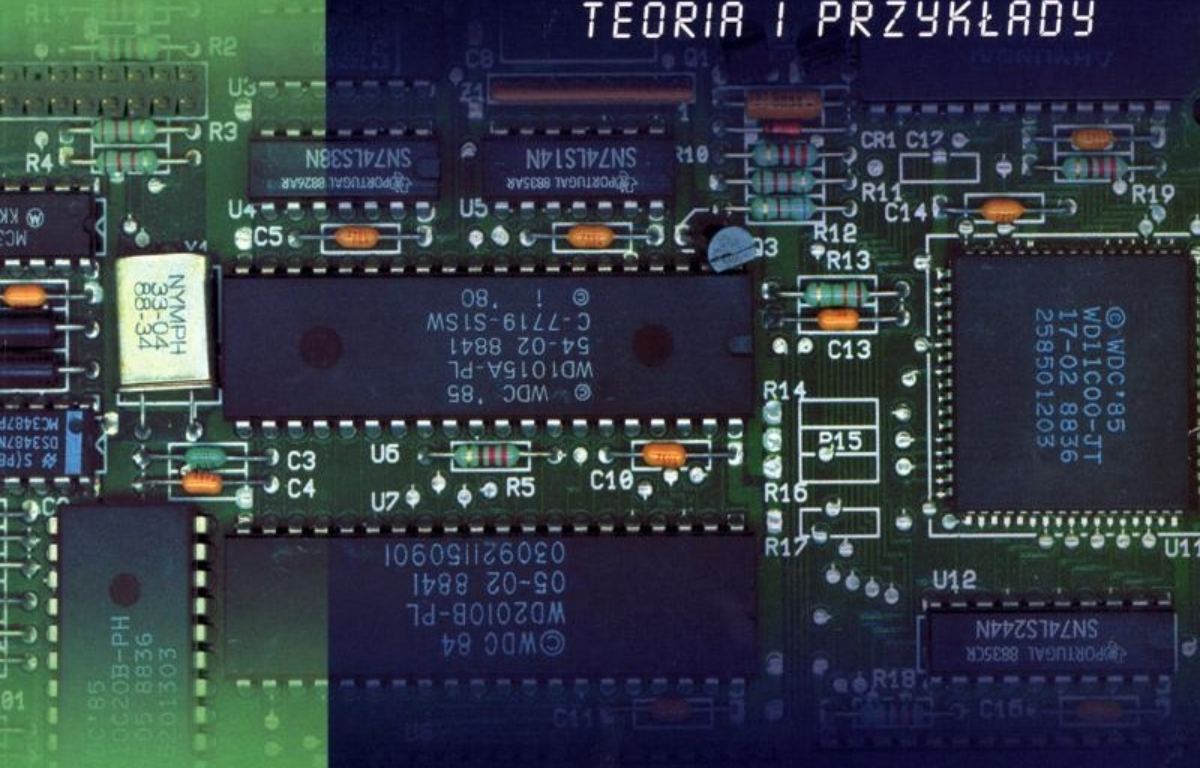
Halina Kamionka-Mikuła
Henryk Małysiak
Bolesław Pochopień

UKŁADY CYFROWE

TEORIA I PRZYKŁADY



WYDAWNICTWO
PRACOWNI KOMPUTEROWEJ JACKA SKALMIERSKIEGO

SPIS TREŚCI

strona

1. PODSTAWY ARYTMETYKI UKŁADÓW CYFROWYCH

Tematyka tego rozdziału obejmuje wybrane zagadnienia związane z zasadami przedstawiania liczb za pomocą umownych symboli (cyfr) i reguł wykonywania na nich działań arytmetycznych.

Wśród systemów liczbowych oprócz systemów niepozycyjnych (np. system rzymski) wyróżniają się systemy pozycyjne, w których znaczenie cyfry jest zależne od miejsca (pozycji) jakie ona zajmuje.

W pozycyjnym systemie liczbowym, (n+m) – pozycyjną nieujemną liczbę

$$X = x_n p^n + \ldots + x_1 p^1 + x_0 p^0 + x_{-1} p^{-1} + \ldots + x_{-m} p^{-m} = \sum_{i=-m}^{n-1} x_i p^i$$

zapisuje się w postaci:

$(X)_p = (x_{n-1} \ldots x_1 x_0 x_{-1} \ldots x_{-m})_p$

przy czym:

p- podstawa systemu liczbowego, $p \in \{2,3,\ldots\}$;

x_i – cyfra i-tej pozycji, $x_i \in \{0,1,\ldots p-1\}$;

n – ilość cyfr części ułamkowej liczby X;

m- ilość cyfr części ułamkowej liczby X;

przecinek - oddziela część całkowitą liczby X od jej części ułamkowej.

Podstawą pozycyjnego systemu liczbowego odpowiada ilość cyfr wykorzystywanych do przedstawienia liczb w tym systemie.

Wartość maksymalnej liczby X, jaką można zapisać w przedstawionej postaci, uzyska się podstawiając: $a_i = p-1$, i wtedy:

$$X_{max} = \sum_{i=-m}^{n-1} (p-1)p^i = p^n - p^{-m}$$

Natomiast wartość minimalna liczby X, różnej od zera, wynosi: $X_{min} = p^{-m}$

Ilość różnych liczb tej postaci można określić z zależności: $\frac{X_{max}}{X_{min}} + 1 = p^{n+m}$

Bezwzględny błąd przedstawienia liczb tej postaci jest definiowany jako minimalna różnica dwóch różnych liczb w tym systemie

$$\Delta = \min | X_1 - X_2 | = p^{-m} \qquad \text{dla} \qquad X_1 \neq X_2$$

W układach cyfrowych najbardziej powszechne uznanie znalazły następujące pozycyjne systemy liczbowe:

- **system dwójkowy (binarny):** $p=2$, $x_i \in \{0,1\}$;
- **system dziesiętny (dziesiątkowy, decymalny):** $p=10$, $x_i \in \{0,1,\ldots,9\}$;
- **system szesnastkowy (heksadecymalny, heksagonalny):** $p=16$, $x_i \in \{0,1,\ldots,9,A,B,C,D,E,F\}$,

przy czym: $(A)_{16} = (10)_{10}$, $(B)_{16} = (11)_{10}$, ... $(F)_{16} = (15)_{10}$

Specyficznym systemem liczbowym jest **system dziesiętny kodowany dwójkowo**, w którym cyfry dziesiętne są przedstawione w kodzie dwójkowym (kodowane dwójkowo).

Dla jednoznacznego przedstawienia dziesięciu cyfr (0,1,....9) systemu dziesiętnego za pomocą zestawu symboli zerojedynkowych trzeba zastosować kod dwójkowy przynajmniej 4-pozycyjny (4 bitowy). Kody służące do kodowania dwójkowego cyfr systemu dziesiętnego noszą nazwę **kodów dwójkowo-dziesiętnych**.
W tabeli 1-1. podano kilka 4-bitowych kodów dwójkowo-dziesiętnych.

Tabela 1-1

Cyfra dziesiętna D	Kody BCD (ang. Binary Coded Decimal)		
	8424	84-2-1	EXCESS-3
0	0000	0000	0011
1	0001	0111	0100
2	0010	0110	0101
3	0011	0101	0110
4	0100	0100	0111
5	0101	1011	1000
6	0110	1010	1001
7	0111	1001	1010
8	1000	1000	1011
9	1001	1111	1100

Każdemu bitowi k_i przypisana jest określona waga w_i (kody wagowe). Wagi mogą być dodatnie lub mieszane (dodatnie i ujemne). Cyfrę dziesiętną przedstawioną za pomocą 4-bitowego kodu dwójkowo-dziesiętnego wagowego można określić z zależności:

$$D = \sum_{i=1}^{4} k_i w_i$$

Kod EXCESS-3 można uważać za kod wagowy, przesunięty o 3 w stosunku do naturalnego kodu dwójkowego. Kod 84-2-1 oraz kod EXCESS-3 mają własność **samouzupełniania**, istotną przy realizacji działań arytmetycznych. Można zauważyć, że jeśli w kodzie 84-2-1 zanegować bity na pozycjach o wagach (-2) i (-1), to uzyska się kod EXCESS-3.

Nieujemna (n+m) –pozycyjna liczba dziesiętna X będzie przedstawiona w systemie dziesiętnym kodowanym dwójkowo w postaci:

$$X_{BCD} = {}^{X}T_{n-1}\,{}^{X}T_{n-2}\ldots{}^{X}T_{1}\,{}^{X}T_{0},\,{}^{X}T_{-1}\ldots{}^{X}T_{-m}$$

przy czym:

$T_i = (x_{i3}\, x_{i2}\, x_{i1}\, x_{i0})$ – i-ta **tetrada**, $\quad i = n\text{-}1, \ldots .1,0,\text{-}1, \ldots .\text{-}m, \quad x_{ij} \in \{0,1\}$.

Przykład 1.1

Liczbę dziesiętną X =459 można przedstawić w systemie dziesiętnym kodowanym dwójkowo w następujący sposób:

- w kodzie 8421: $X_{8421} = 0100\ \ 0101\ \ 1001$
- w kodzie EXCESS-3: $X_{EX\text{-}3} = 0111\ \ 1000\ \ 1100$

W systemie liczbowym o podstawie p operacje arytmetyczne dodawania, odejmowania i mnożenia dwóch (n+m) –pozycyjnych nieujemnych liczb:

$$(X)_p = (x_{n-1} \ldots x_{i+1}\ x_i\ x_{i-1} \ldots x_{-m})_p$$
$$(Y)_p = (y_{n-1} \ldots y_{i+1}\ y_i\ y_{i-1} \ldots y_{-m})_p$$

można sprowadzić do wykonania tych operacji na pojedynczych cyfrach tych liczb.

Rezultatem wykonania operacji na cyfrach x_i i y_i w systemie o podstawie p są dwie wartości:

s_i - wynik operacji na i-tej pozycji;

r_{i+1} - przeniesienie w kierunku (i+1) –szej pozycji (dla dodawania i mnożenia) lub pożyczka z (i+1) –szej pozycji (dla odejmowania).

Zasady uzyskania wartości s i r można sformułować następująco:

dodawanie

$$(s_i)_d = \begin{cases} x_i + y_i + r_i & \text{dla } x_i + y_i + r_i < p \\ x_i + y_i + r_i - p & \text{dla } x_i + y_i + r_i \geq p \end{cases}$$

$$(r_{i+1})_d = \begin{cases} 0 & \text{dla } x_i + y_i + r_i < p \\ 1 & \text{dla } x_i + y_i + r_i \geq p \end{cases}$$

odejmowanie

$$(s_i)_o = \begin{cases} x_i - y_i - r_i & \text{dla } x_i - y_i - r_i \geq 0 \\ x_i - y_i - r_i + p & \text{dla } x_i - y_i - r_i < 0 \end{cases}$$

$$(r_{i+1})_o = \begin{cases} 0 & \text{dla } x_i - y_i - r_i \geq 0 \\ 1 & \text{dla } x_i - y_i - r_i < 0 \end{cases}$$

mnożenie

$$(s_i)_m = \begin{cases} x_i y_i + r_i & \text{dla } x_i y_i + r_i < p \\ x_i y_i + r_i - \left[\dfrac{x_i y_i + r_i}{p}\right] p & \text{dla } x_i y_i + r_i \geq p \end{cases}$$

$$(r_{i+1})_d = \begin{cases} 0 & \text{dla } x_i y_i + r_i < p \\ \left[\dfrac{x_i y_i + r_i}{p}\right] & \text{dla } x_i y_i + r_i \geq p \end{cases}$$

[1] [a] = entier (a) – część całkowita liczby a

W systemie dwójkowym mnożenie znacznie się upraszcza, bowiem zawsze $(r_{i+1})_m = 0$. Dla systemu dwójkowego uzyskuje się wartości s i r, tak jak to pokazano w tabeli 1-2.

Tabela 1-2

x y	x + y	x - y	x y
	s_d r_d	s_o r_o	s_m
0 0	0 0	0 0	0
0 1	1 0	1 1	0
1 0	1 0	1 0	0
1 1	0 1	0 0	1

Realizacja operacji dzielenia dwóch liczb nie może być sprowadzona do działań na pojedynczych cyfrach. W przypadku wykonywania operacji na cyfrach dziesiętnych kodowanych dwójkowo należy uwzględnić dodatkowo konsekwencje wynikające z przyjętego kodowania.

W przypadku dodawania dwóch i-tych cyfr dziesiętnych kodowanych dwójkowo (${}^{X}T_i$, ${}^{Y}T_i$) oraz ewentualnego przeniesienia C_i z (i-1)–szej tetrady liczb:

$$X_{BCD} = {}^{X}T_{n-1}\ {}^{X}T_{n-2} \ldots {}^{X}T_i \ldots$$
$$Y_{BCD} = {}^{Y}T_{n-1}\ {}^{Y}T_{n-2} \ldots {}^{Y}T_i \ldots$$

W wyniku tego dodawania

$${}^{X}T_i + {}^{Y}T_i + C_i$$

można uzyskać wartości odpowiadające 0,1,2,....18 lub 19.

Zestawmy w postaci tabel: 1-3 oraz 1-4 uzyskane wartości z dodawania odpowiedników dwójkowych cyfr (T') oraz prawidłowe ich wartości w określonym kodzie (T) dla kodów:

- 8421 (tabela 1-3.)
- EXCESS-3 (tabela 1-4.)

Z przedstawionych tabel wynika, że zasady dodawania są uzależnione od kodu, a w niektórych przypadkach wymagana jest korekta uzyskanego wyniku.

Tabela 1-3

Suma dziesiętna	$({}^{X}T_i + {}^{Y}T_i + C_i)_{8421}$		Spostrzeżenia
	T'	T	
0	0000	0000	
1	0001	0001	
2	0010	0010	
3	0011	0011	
4	0100	0100	T = T'
5	0101	0101	
6	0110	0110	
7	0111	0111	
8	1000	1000	
9	1001	1001	
10	1010	1←0000	
11	1011	1←0001	
12	1100	1←0010	
13	1101	1←0011	
14	1110	1←0100	
15	1111	1←0101	T = T' + 0110
16	1 0000	1←0110	
17	1 0001	1←0111	
18	1 0010	1←1000	
19	1 0011	1←1001	

Tabela 1-4

Suma dziesiętna	$({}^{X}T_i + {}^{Y}T_i + C_i)_{EX\text{-}3}$		Spostrzeżenia
	T'	T	
0	0110	0011	
1	0111	0100	T = T' – 0011
2	1000	0101	lub
3	1001	0110	T = T' + 1101
4	1010	0111	
5	1011	1000	(z pominięciem
6	1100	1001	przeniesienia
7	1101	1010	z tetrady)
8	1110	1011	
9	1111	1100	
10	1 0000	1←0011	
11	1 0001	1←0100	
12	1 0010	1←0101	
13	1 0011	1←0110	
14	1 0100	1←0111	
15	1 0101	1←1000	T = T' + 0011
16	1 0110	1←1001	
17	1 0111	1←1010	
18	1 1000	1←1011	
19	1 1001	1←1100	

Analiza tych tablic upoważnia do sformułowania zasad dodawania cyfr liczb dziesiętnych kodowanych dwójkowo w sposób następujący:

Kod 8421

- dla $0000 \leq T' \leq 1001$ ← jest $T = T'$ (brak korekty);
- dla $1010 \leq T' \leq 10011$ ← jest $T = T' + 0110$

Okazuje się, że wprowadzenie poprawek (korekta) może być wielostopniowe.

Nie można w tym przypadku uzależnić wprowadzenia poprawek od wystąpienia przeniesienia.

Kod EXCESS-3

- dla $0110 \leq T' \leq 1111$ ← jest $T = T' - 0011$
  lub $T = T' + 1101$ (przy czym występujące przeniesienie z tetrady należy pominąć):
- dla $10000 \leq T' \leq 11001$ ← jest $T = T' + 0011$.

W tym przypadku wartość wprowadzonej poprawki można uzależnić od przeniesienia C_{i+1}:

- jeśli = 0, to $T = T' - 0011$ lub $T = T' + 1101$ (przeniesienie z tetrady pominąć)
- jeśli =1, to $T = T' + 0011$.

Wprowadzenie poprawek w tym przypadku jest zawsze jednostopniowe.

Przykład 1.2

Wykonajmy dodawanie dwóch liczb dziesiętnych $(469)_{10}$ i $(578)_{10}$:

- w systemie dziesiętnym

$$\begin{array}{rl} (X)_{10} = & 469 \\ (Y)_{10} = & 578 \\ \hline (Z)_{10} = & 1047 \end{array} \Big\} +$$

- w systemie dziesiętnym o cyfrach w kodzie 8421

$$\begin{array}{rcccc} X_{8421} = & 0000 & 0100 & 0110 & 1001 \\ Y_{8421} = & 0000 & 0101 & 0111 & 1000 \\ \hline & 0000 & 1001 & 1110 \leftarrow & 0001 \\ + & 0000 & 0000 & 0110 & 0110 \quad \text{(poprawka)} \\ \hline & 0000 & 1010 \leftarrow & 0100 & 0111 \quad \text{(poprawka)} \\ + & 0000 & 0110 & 0000 & 0000 \\ \hline Z_{8421} = & 0001 & 0000 & 0100 & 0111 \\ & \underbrace{\quad} & \underbrace{\quad} & \underbrace{\quad} & \underbrace{\quad} \\ & 1 & 0 & 4 & 8 \end{array} \Big\} + \quad = (Z)_{10}$$

- w systemie dziesiętnym o cyfrach w kodzie EXCESS-3

$X_{EX-3} = 0011\ 0111\ 1001\ 1100$
$Y_{EX-3} = 0011\ 1000\ 1010\ 1011$ } +

$0111 \leftarrow 0000 \leftarrow 0100 \leftarrow 0111$
$+\ 1101\ 0011\ 0011\ 0011$ (poprawka)

$Z_{EX-3} = 0100\ 0011\ 0111\ 1010$
$\not{1}\ \ 1\ \ 0\ \ 4\ \ 7$ $(Z)_{10}$

(pomija się przeniesienie)

- w systemie dwójkowym

$(X)_2 = 0\,1\,1\,1\,0\,1\,0\,1\,0\,1$
$(Y)_2 = 1\,0\,0\,1\,0\,0\,0\,0\,1\,0$ } +

$(Z)_2 = 1\,0\,0\,0\,0\,0\,1\,0\,1\,1\,1$

- w systemie szestnastkowym

Znajdźmy najpierw odpowiedniki liczb X i Y w systemie szesnastkowym.

$(X)_2 = 0111\ 0101$
$1\ \ D\ \ 5 = (X)_{16}$

$(Y)_2 = 1001\ 0000\ 1010$
$2\ \ 4\ \ 2 = (Y)_{16}$

$(X)_{16} = 1\ D\ 5$
$(Y)_{16} = 2\ 4\ 2$ } +

$(Z)_{16} = 4\ 1\ 7$

Warto zwrócić uwagę na dodawanie dwóch cyfr środkowych tych liczb, a mianowicie:

$$(D)_{16} + (4)_{16} = \underbrace{(D)_{16} + (3)_{16}}_{(10)_{16}} + (1)_{16} = (10)_{16} + (1)_{16} = (11)_{16}$$

Łatwo sprawdzić poprawność wykonanego dodawania $(X)_{16} + (Y)_{16}$, gdyż

$$(X)_{16} + (Y)_{16} = (Z)_{16} = \underbrace{4}_{100}\ \underbrace{1}_{0001}\ \underbrace{7}_{0111} = (X)_2$$

Z praktycznego punktu widzenia system liczbowy powinien charakteryzować się m.in. możliwością prostego i jednoznacznego przedstawiania liczb z interesującego zakresu, prostotą wykonywania działań na liczbach, łatwością technicznej realizacji układu cyfrowego wykonującego działania na liczbach tego systemu.

We współczesnych układach cyfrowych stosowany jest przede wszystkim dwójkowy system liczbowy. Wynika to głównie z łatwości konstrukcji elementów mogących znajdować się w dwóch stanach, którym mogą odpowiadać wartości cyfr dwójkowego systemu liczbowego. Porównania systemu dwójkowego i dziesiętnego kodowanego 4-bitowym kodem dwójkowym wskazuje na korzyść tego pierwszego, gdyż liczba bitów potrzebnych do przedstawienia liczby dziesiętnej wymaga 0,2 krotnego zwiększenia liczby bitów w stosunku do ilości bitów jej odpowiednika dwójkowego. Mimo to, często stosuje się przedstawienie liczb w postaci dziesiętnej kodowanej dwójkowo, a w szczególności przy stosunkowo prostych na nich działaniach i wtedy gdy straty związane z czasem ich zamiany na postać dwójkową są większe niż korzyść z przetwarzania dwójkowego.

Okazuje się, że operację odejmowania arytmetycznego można realizować z zastosowaniem operacji dodawania arytmetycznego. Wymaga to jednak wykorzystania pojęcia uzupełnienia liczby.

W pozycyjnym systemie liczbowym o podstawie p dla (n+m) –pozycyjnej liczby nieujemnej

$$X = x_{n-1}\, x_{n-2} \ldots x_1 x_0,\ x_{-1} \ldots x_{-m}$$

definiuje się dwa rodzaje uzupełnień:

- **uzupełnienie p-te**:

$$\overline{\overline{X}} = p^n - X \quad \text{dla} \quad A \neq 0$$

$$\overline{\overline{X}} = 0 \quad \text{dla} \quad X = 0$$

- **uzupełnienie (p-1)-sze:**

$$\overline{X} = p^n - X - p^{-m}$$

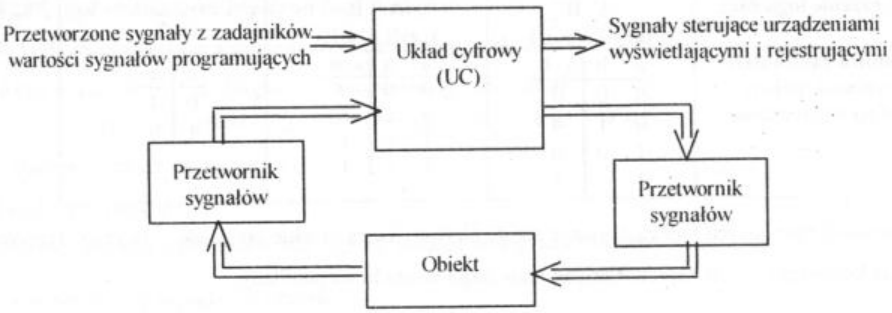
W zależności od wartości podstawy systemu liczbowego uzupełnienia przyjmują nazwy: uzupełnienie dwójkowe i jedynkowe (p=2), uzupełnienie dziesiątkowe i dziewiątkowe (p=10), uzupełnienie szesnastkowe i piętnastkowe (p=16).

Wtedy operacja odejmowania dwóch nieujemnych liczb A i B , o jednakowych podstawach, może być wykonywana przez dodanie do odjemnej p-tego lub (p-1)-szego uzupełnienia odjemnika, z zastosowaniem określonych zasad.

Z kolei operacje arytmetycznego mnożenia i dzielenia mogą być wykonywane z zastosowaniem operacji arytmetycznego dodawania oraz operacji przesuwania.

2. PODSTAWOWE POJĘCIA

Układ cyfrowy (logiczny, przełączający) - wykonuje operacje na zbiorach sygnałów dyskretnych, najczęściej dwuwartościowych. W oparciu o przetworzone sygnały programujące i przetworzone sygnały pochodzące z czujników obiektu, wypracowuje on (rys. 2-1) sygnały sterujące obiektem po ich przetworzeniu i sygnały sterujące urządzeniami wskazującymi i rejestrującymi. Obiektem może być np. obiekt przemysłowy lub jednostka arytmetyczno-logiczna (JAL) procesora.



Rys. 2-1

Działanie układu przełączającego można opisać w postaci zdań orzekających zgodnie z regułami logiki matematycznej. W logice dwuwartościowej stwierdzenia, które muszą być prawdą lub fałszem, nazywamy ***zdaniami logicznymi***. Przykład zdania logicznego Z: *temperatura jest wyższa od wartości* t_1 i zdania przeciwnego do Z (negacji logicznej zdania Z, czyli $\overline{Z}$): *temperatura nie jest wyższa od* t_1. Zdania logiczne połączone spójnikiem „i" tworzą *iloczyn logiczny* tych zdań. Zdania logiczne połączone spójnikiem „lub" tworzą *sumę logiczną* tych zdań.

Teoria Automatów zajmuje się metodami syntezy i analizy układów cyfrowych. Do ich opisu wykorzystuje ona algebrę Boole'a.

2.1. Algebra Boole'a

Jest ona sformalizowanym uogólnieniem rachunku logicznego zdań zgodnie z odpowiednikami:

rachunek log. zdań	**algebra Boole'a**
prawda	1
fałsz	0
zdanie logiczne A	A
zdanie przeciwne do A	$\overline{A}$
iloczyn logiczny zdań: A,B	$A \cdot B$
suma logiczna zdań: A,B	$A + B$

Algebra Boole'a obejmuje działania, w których każdy z argumentów, jak i wyniki działania mogą przyjmować tylko dwie wartości: 0 lub 1. Funkcje, którymi operuje algebra Boole'a nazywamy funkcjami przełączającymi (logicznymi).

Podstawowe funkcje algebry Boole'a przedstawia tabela 2-1.

Tabela 2-1

Nazwa funkcji:	Iloczyn logiczny (koniunkcja)	Suma logiczna (dysjunkcja)	Negacja logiczna (inwersja)
Wyrażenie logiczne:	$A \cdot B$ (czyli $A \wedge B$)	$A + B$ (czyli $A \vee B$)	$\overline{A}$
Tablica zależności: wartości funkcji od jej argumentów	a b | a·b 0 0 | 0 0 1 | 0 1 0 | 0 1 1 | 1	a b | a+b 0 0 | 0 0 1 | 1 1 0 | 1 1 1 | 1	a | $\overline{a}$ 0 | 1 1 | 0

W wyrażeniach opisujących funkcje logiczne reguły pierwszeństwa są takie same, jak w zwykłej algebrze. Znak mnożenia logicznego: „·" w zapisie iloczynu logicznego często bywa pomijany.

Podstawowe prawa algebry Boole'a - zawarte są w tabeli 2-2.

Tabela 2-2

1. Prawo przemienności	dla mnożenia ($\wedge$):	$A \cdot B = B \cdot A$
	dla dodawania ($\vee$):	$A + B = B + A$
2. Prawo łączności	dla $\wedge$:	$(A \cdot B) \cdot C = A \cdot (B \cdot C)$
	dla $\vee$:	$(A + B) + C = A + (B + C)$
3. Prawo rozdzielności	dla $\wedge$ względem $\vee$:	$A \cdot (B + C) = A \cdot B + A \cdot C$
	dla $\vee$ względem $\wedge$:	$A + (B \cdot C) = (A + B) \cdot (A + C)$
4. Prawo negacji (de Morgana)	dla $\wedge$:	$\overline{A \cdot B} = \overline{A} + \overline{B}$
	dla $\vee$:	$\overline{A + B} = \overline{A} \cdot \overline{B}$

Inne zależności wynikające z podstawowych funkcji i praw algebry Boole'a podaje tabela 2-3.

Tabela 2-3

$0 \cdot 1 = 0$	$0 + 1 = 1$
$0 \cdot A = 0$	$0 + A = A$
$1 \cdot A = A$	$1 + A = 1$
$A \cdot A = A$	$A + A = A$
$\overline{A} \cdot A = 0$	$A + \overline{A} = 1$
$\overline{\overline{A}} = A$	

Pojęcia stosowane w zapisie funkcji logicznych

Składnik "1", czyli elementarny implikant (ek) funkcji F

Jest to kombinacja wartości argumentów, dla których wartość funkcji F wynosi 1;

Może być wyrażony: dwójkowo, dziesiętnie, lub literałowo.

Np.: $(10011)_{abcde} \equiv (19)_{abcde} \equiv a \cdot \bar{b} \cdot \bar{c} \cdot d \cdot e$.

F^1 - zbiór wszystkich ek funkcji F

Czynnik „0", czyli elementarny implicent (ec) funkcji F

Jest to kombinacja wartości argumentów, dla których wartość funkcji F wynosi 0;

Może być wyrażony: dwójkowo, dziesiętnie, lub literałowo.

Np.: $(10011)_{abcde} \equiv (19)_{abcde} \equiv \bar{a} + b + c + \bar{d} + \bar{e}$.

F^0 - zbiór wszystkich ec funkcji F.

Stan Φ, czyli stan obojętny

Kombinacja wartości argumentów, dla których wartość funkcji jest dowolna.

Literał - argument lub negacja argumentu.

Implikant

Jest to funkcja Ik, dla której spełniona jest zależność: jeżeli dla pewnej kombinacji wartości argumentów funkcji F implikant Ik = *1*, to dla tej wartości argumentów F = *1*.

Implicent

Jest to funkcja Ic, dla której spełniona jest zależność: jeżeli dla pewnej kombinacji wartości argumentów funkcji F implicent Ic = *0*, to dla tej wartości argumentów F = *0*.

Implikant prosty

Jest to elementarny iloczyn (iloczyn pojedynczych literałów) będący implikantem funkcji F, który zredukowany o dowolny literał przestaje być implikantem funkcji F.

Implicent prosty

Jest to elementarna suma (suma pojedynczych literałów) będący implicentem funkcji F, który zredukowany o dowolny literał przestaje być implicentem funkcji F.

Postać sumy (alternatywna, dysjunkcyjna) funkcji F

Suma *składników (implikantów)* tej funkcji, gdzie składnik jest iloczynem pojedynczych literałów.

Postać iloczynu (koniunkcyjna) funkcji F

Iloczyn *czynników (implicentów)* tej funkcji, gdzie czynnik jest sumą pojedynczych literałów.

Kanoniczna postać sumy funkcji F (nieuproszczona) - to suma wszystkich *składników "1"* funkcji F.

Kanoniczna postać iloczynu funkcji F (nieuproszczona) - to iloczyn wszystkich *czynników "0"* funkcji F.

Quasi-elementarny implikant (qk) dla F

Implikant, który pochłania parę sąsiednich logicznie ek funkcji F lub taki pojedynczy składnik "*1*" tej funkcji, który nie posiada swojego logicznego "sąsiada".

Qk jest zbiorem wszystkich qk danej funkcji. Qk′ jest podzbiorem zbioru Qk zawierającym wszystkie pary sąsiednich logicznie ek (bez pojedynczych ek).

Quasi-elementarny implicent (qc) dla F

Implicent, który pochłania parę sąsiednich logicznie ec funkcji F lub taki pojedynczy *czynnik* "*0*" tej funkcji, który nie posiada swojego logicznego "sąsiada".

Qc jest zbiorem wszystkich qc danej funkcji. Qc′ jest podzbiorem zbioru Qc zawierającym wszystkie pary sąsiednich logicznie ec (bez pojedynczych ec).

Postać skrócona funkcji logicznej

Charakteryzuje się tym, że w porównaniu do kanonicznej zawiera mniej literałów. Postać skrócona może ale nie musi być postacią minimalną.

Postać nieskracalna (nieredundancyjna)

Nie można z niej usunąć żadnego literału bez zmiany funkcji.

Postać minimalna funkcji

To zapis funkcji, zawierający najmniejszą liczbę literałów i działań logicznych.

Postać skrócona sumy lub iloczynu

Zawiera mniej literałów niż postać kanoniczna.

Normalna postać sumy - postać sumy nie zawierająca składników tożsamościowo równych „*0*" np.: $a \cdot \overline{a} \cdot b \cdot c$

Normalna postać iloczynu - postać iloczynu nie zawierająca czynników tożsamościowo równych „*1*". np.:

$$a + \overline{a} + b + \overline{c}$$

Wyrażenie normalne - jest to wyrażenie o normalnej postaci sumy lub normalnej postaci iloczynu.

Wyrażenie różne od normalnego

Przyjęto, że jest to wyrażenie o postaci sumy lub iloczynu, dla sumy zawierające składniki między innymi tożsamościowo równe „*0*" np.: $a \cdot b \cdot \overline{b} \cdot c$ lub odpowiednio dla iloczynu czynniki między innymi tożsamościowo równe „*1*" np.: $\left(a + b + \overline{b} + c\right)$.

Przykład 2.1

W tabeli 2-4 przedstawione są wartości dwóch funkcji logicznych: Z_1 i Z_2. W tabeli 2-5 podane są składniki jedynki (elementarne implikanty) i czynniki zera (elementarne implicenty) dla tych funkcji. Funkcja Z_1 nie posiada stanów Φ, czyli jest *w pełni określona*. Funkcja Z_2 dla kombinacji argumentów: $(111)_{abc}$ nie jest określona, czyli może przyjmować dowolną wartość (wartość Φ). Mówimy, że jest ona *nie w pełni określona*.

Tabela 2-4

a	b	c	Z_1	Z_2
0	0	0	0	0
0	0	1	0	0
0	1	0	1	1
0	1	1	1	1
1	0	0	0	1
1	0	1	1	0
1	1	0	0	1
1	1	1	1	Φ

Tabela 2-5

		zapis binarny	zapis dziesiętny
Z_1	„składniki jedynki":	$(010, 011, 101, 111)_{abc}$	$(2, 3, 5, 7)_{abc}$
	„czynniki zera":	$(000, 001, 100, 110)_{abc}$	$(0, 1, 4, 6)_{abc}$
Z_2	„składniki jedynki":	$(010, 011, 100, 110)_{abc}$	$(2, 3, 4, 6)_{abc}$
	„czynniki zera":	$(000, 001, 101)_{abc}$	$(0, 1, 5)_{abc}$

Oto ilustracje podstawowych pojęć dla Z_1

Kanoniczna postać funkcji:

Zapis dwójkowy wyrażenia o postaci sumy: $Z_1 = \Sigma(010, 011, 101, 111)_{abc}$

Zapis dwójkowy wyrażenia o postaci iloczynu: $Z_1 = \Pi(000, 001, 100, 110)_{abc}$

Zapis dziesiętny wyrażenia o postaci sumy: $Z_1 = \Sigma(2, 3, 5, 7)_{abc}$

Zapis dziesiętny wyrażenia o postaci iloczynu $Z_1 = \Pi(0, 1, 4, 6)_{abc}$

Zapis literałowy wyrażenia o postaci sumy: $Z_1 = \bar{a} \cdot b \cdot \bar{c} + \bar{a} \cdot b \cdot c + a \cdot \bar{b} \cdot c + a \cdot b \cdot c$

Zapis literałowy wyrażenia o postaci iloczynu: $Z_1 = (a + b + c)(a + b + \bar{c})(\bar{a} + b + c)(\bar{a} + \bar{b} + c)$

Skrócona postaci funkcji: $Z_1 = \bar{a}\, b + a\, c + b\, c$

Nieskracalna (nieredundancyjna) postać funkcji: $Z_1 = \bar{a}\, b + a\, c$

Niektóre sposoby przedstawienia kanonicznej postaci funkcji nie w pełni określonej na przykładzie funkcji Z_2 w zapisie dziesiętnym:

sposób 1:

$$\begin{cases} Z_2 = \Sigma(2, 3, 4, 6)_{abc} \\ Z_2 = \Pi(0, 1, 5)_{abc} \end{cases}$$

sposób 2:

$$Z_2 = \Sigma[2, 3, 4, 6\ (7)]_{abc}$$

sposób 3

$$Z_2 = \Pi[0, 1, 5\ (7)]_{abc}$$

Podział układów przełączających

Rys. 2-2

Układem kombinacyjnym nazywamy układ, w którym kombinacje wartości sygnałów wejściowych w sposób jednoznaczny określają kombinacje wartości sygnałów wyjściowych, czyli każdemu stanowi wejść odpowiada ściśle określony stan wyjść.

Układem sekwencyjnym jest układ, w którym istnieje przynajmniej jeden taki stan wejść, któremu odpowiada kilka różnych stanów wyjść. To, który ze stanów wyjść przy zadanym stanie wejść pojawi się w danej chwili na wyjściu, zależy nie tylko od aktualnego stanu wejść, ale także od kombinacji poprzednich sygnałów wejściowych.

W układach sekwencyjnych musi występować *Blok Pamięci (BP)*, który pamięta kolejność zmian sygnałów wejściowych. Sygnały wyjściowe bloku pamięci określają tzw. *wewnętrzny stan układu*. Układy sekwencyjne można podzielić na synchroniczne i asynchroniczne.

W **układach sekwencyjnych synchronicznych** zmiany stanów wewnętrznych układu mogą występować tylko w ściśle określonych chwilach czasu, wyznaczonych przez dodatkowy sygnał taktujący (zegarowy, taktujący, synchronizujący).

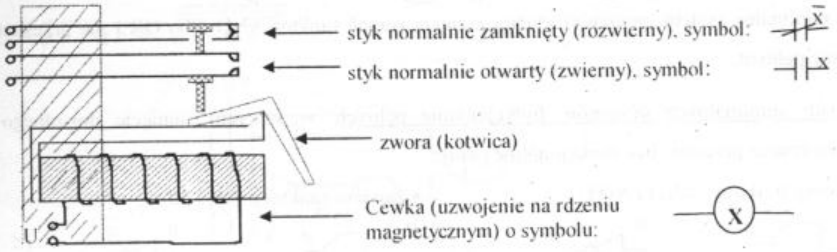
W **układach sekwencyjnych asynchronicznych** zmiany stanów wewnętrznych układu mogą występować w dowolnych chwilach czasu, określonych przez zmiany sygnałów wejściowych układu. Układy te dzieli się na:

- statyczne (potencjałowe), w których informacja przenoszona jest wyłącznie w postaci poziomów sygnałów logicznych oznaczonych przez „0" i „1", czyli sygnałów statycznych,
- dynamiczne (potencjałowo-impulsowe), w których informacja może być przenoszona zarówno w postaci statycznych sygnałów („0" i „1"), jak i impulsowych sygnałów pojawiających się przy zmianach wartości sygnałów statycznych: 0→1 lub 1→0.

3. ELEMENTY PRZEŁĄCZAJĄCE I OZNACZENIA

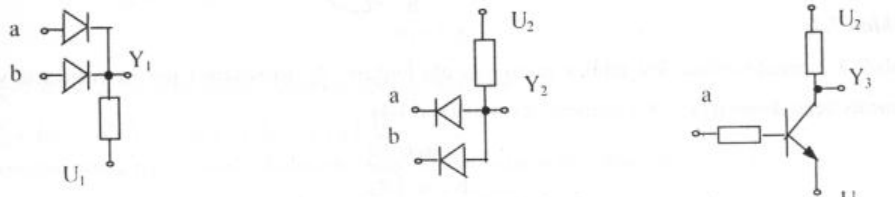
Do realizacji układów przełączających (układów cyfrowych) są używane elementy logiczne stykowe i bezstykowe wykorzystujące różne zjawiska fizyczne. Do najbardziej rozpowszechnionych należą elementy: np.: elektroniczne, elektromagnetyczne, hydrauliczne, pneumatyczne, magnetyczne, światłowodowe.

Przykład elementu stykowego, jakim jest przekaźnik elektromagnetyczny X przedstawia rys. 3-1.



Rys. 3-1

Przykłady rozwiązań najprostszych półprzewodnikowych układów logicznych (funktorów) przedstawia rysunek 3-2.



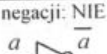
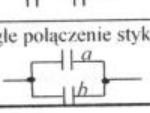
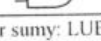
$Y_1 = a+b$ (konw. dodatnia)
$Y_1 = a \cdot b$ (konw. ujemna)

$Y_2 = a \cdot b$ (konw. dodatnia)
$Y_2 = a+b$ (konw. ujemna)

$Y_3 = \overline{a}$

Rys. 3-2

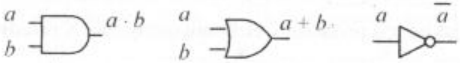
Tabela 3-1

algebra Boole'a	układy stykowe	układy bezstykowe
0	przerwa	Poziom napięcia $U_1 \mp \Delta U$ — w konwencji dodatniej $U_1 < U_2$
1	zwarcie	Poziom napięcia $U_2 \mp \Delta U$ — w konwencji ujemnej $U_1 > U_2$
a	styk normalnie otwarty (n.o.) 	„szyna" z sygnałem a 
$\overline{a}$	styk normalnie zamknięty (n. z.) 	funktor negacji: NIE (NOT) 
$a \cdot b$	szeregowe połączenie styków n. o. 	funktor iloczynu: I (AND) 
$a + b$	równoległe połączenie styków n. o. 	funktor sumy: LUB (OR) 

System funkcjonalnie pełny

Jest nim taki zestaw funktorów, który pozwala zrealizować każdą funkcję logiczną. Przykładowe systemy funkcjonalnie pełne podaje zestawienie.

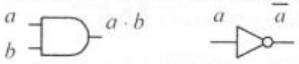
Podstawowy system funkcjonalnie pełny zawiera o funktory: AND, OR, NOT:



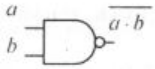
Nie jest to minimalny system, ponieważ można z niego usunąć funktor AND albo OR i nie przestanie on być funkcjonalnie pełnym.

Oto przykłady minimalnych systemów funkcjonalnie pełnych, w których usunięcie dowolnego funktora powoduje, że system przestaje być funkcjonalnie pełny:

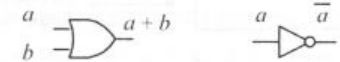
- zawiera funktory: AND i NOT:



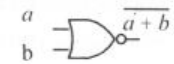
- zawiera funktory: NAND:



- zawiera funktory: OR i NOT:



- zawiera funktory: NOR:



Przykład 3.1

W tabeli 3-2 przedstawiona jest tablica zależności dla funkcji Z_3. Rozważana jest jej realizacja w oparciu o elementy bezstykowe (rys. 3-3) i elementy stykowe (rys. 3-4).

Tabela 3-2

a	b	c	Z_3
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

Kanoniczna postać sumy rozważanej funkcji jest następująca:

$$Z_3 = \bar{a}\cdot\bar{b}\cdot\bar{c} + \bar{a}\cdot\bar{b}\cdot c + a\cdot\bar{b}\cdot\bar{c} + a\cdot b\cdot\bar{c}$$

Po zastosowaniu prawa rozdzielności mnożenia względem dodawania otrzymujemy:

$$Z_3 = \bar{a}\cdot\bar{b}\cdot(\bar{c} + c) + a\cdot\bar{c}(\bar{b} + b)$$ a następnie postać minimalną sumy: $Z_3 = \bar{a}\,\bar{b} + a\,\bar{c}$.

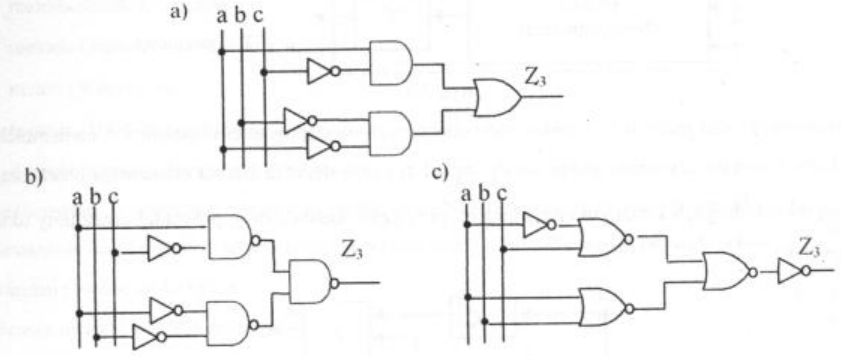
Realizację minimalnej postaci sumy dla Z_3 w oparciu o systemy funkcjonalnie pełne przedstawia rys. 3-3: a –system podstawowy, b - system zawierający funktory NAND, c) - system zawierający funktory NOR.

Dla realizacji funkcji w oparciu o elementy (funktory) NAND potrzebne jest przekształcenie wyrażenia (korzystając z prawa negacji) do postaci zawierającej wyłącznie operacje negacji iloczynu literałów.

$$Z_3 = \bar{a}\,\bar{b} + a\,\bar{c} = \overline{\overline{\bar{a}\,\bar{b}}\;\;\overline{a\,\bar{c}}}$$

Podobnie dla realizacji funkcji w oparciu o elementy (funktory) NOR potrzebne jest przekształcenie wyrażenia (korzystając z prawa negacji) do postaci zawierającej wyłącznie operacje negacji sumy literałów.

$$Z_3 = \bar{a}\,\bar{b} + a\,\bar{c} = \overline{\overline{a+b} + \overline{\bar{a}+c}}$$



Rys. 3-3

Dla funkcji z tabeli 3-2 można podać kanoniczną postać iloczynu:

$$Z_3 = (a + \bar{b} + c)\cdot(a + \bar{b} + \bar{c})\cdot(\bar{a} + b + \bar{c})\cdot(\bar{a} + \bar{b} + \bar{c}).$$

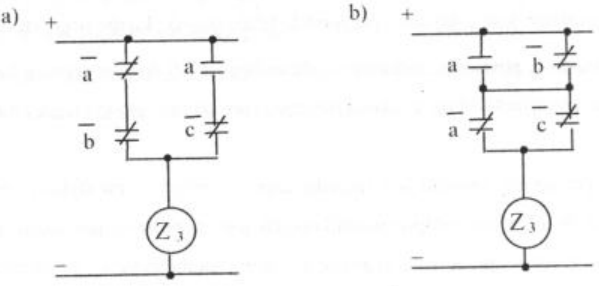
Po zastosowaniu prawa rozdzielności dodawania względem mnożenia otrzymujemy:

$$Z_3 = (a + \bar{b} + c\cdot\bar{c})\cdot(\bar{a} + \bar{c} + b\cdot\bar{b}).$$

a następnie postać minimalną iloczynu: $Z_3 = (a + \bar{b})\cdot(\bar{a} + \bar{c})$.

Realizację funkcji Z_3 w oparciu o elementy stykowe przedstawia rys. 3-4:

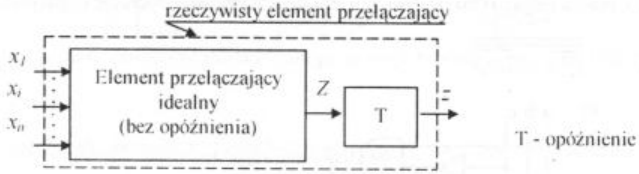
- dla minimalnej postaci sumy rys. 3-4a.
- dla minimalnej postaci iloczynu rys. 3-4b.



Rys. 3-4

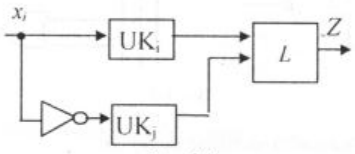
Hazard w układach cyfrowych

Z powodu opóźnień występujących w rzeczywistych elementach przełączających (rys. 3-5), w układach cyfrowych mogą występować chwilowe przeklamania z powodu błędnie realizowanych zależności: $x + \bar{x} = 1$ lub $x \cdot \bar{x} = 0$.



Rys. 3-5

Hazard (podstawowy) - jest jednym ze zjawisk spowodowanych występowaniem opóźnień w elementach przełączających. Mamy z nim do czynienia wtedy, kiedy sygnał x_i i jego negacja sterują elementem przełączającym L po co najmniej dwóch drogach z różnym opóźnieniem, przy czym każda z dróg prowadzi przez inny układ kombinacyjny (rys. 3-6).



Rys. 3-6

Rodzaje hazardu:

`HSd` - hazard statyczny w warunkach działania - może powodować błąd typu $1 \rightarrow 0 \rightarrow 1$, gdy wartość sygnału Z powinna pozostać niezmieniona równa 1,

`HSn` - hazard statyczny w warunkach niedziałania - może powodować błąd typu $0 \rightarrow 1 \rightarrow 0$, gdy wartość sygnału Z powinna pozostać niezmieniona równa 0,

`HD` - hazard dynamiczny - może powodować błąd typu $1 \rightarrow 0 \rightarrow 1 \rightarrow 0$ lub $0 \rightarrow 1 \rightarrow 0 \rightarrow 1$, gdy wartość sygnału Z powinna się zmienić jeden raz odpowiednio: $1 \rightarrow 0$ lub $0 \rightarrow 1$,

`wHSd (wHSn)` - rozumiany jest jako taki przypadek `HSd (HSn)`, kiedy przy zmianie sygnału x_i sygnał wyjściowy elementu L powinien zachować stałą wartość 1 (0), lecz w wyniku hazardu wartość ta może się zmieniać: $1 \rightarrow 0 \rightarrow 1 \rightarrow 0 \rightarrow 1$ $(0 \rightarrow 1 \rightarrow 0 \rightarrow 1 \rightarrow 0)$, czyli może wystąpić zwielokrotnienie błędu typu `HSd (HSn)`.

W układach stykowych przyczyną hazardu jest niejednoczesna zmiana stanu styków: zwiernych i rozwiernych tego samego przekaźnika. W układach półprzewodnikowych jego przyczyną jest różnica opóźnień dróg logicznych, po których zmiany proste i przeciwne sygnału x_i sterują elementem L. W układach z pamięcią oprócz hazardu mogą występować zjawiska: wyścigu i hazardu zasadniczego (dokładniej omówione w rozdziale 15).

4. MINIMALIZACJA WYRAŻEŃ LOGICZNYCH

Celem jej jest określenie ***wyrażenia minimalnego***, czyli zawierającego minimalną liczbę literałów i minimalną liczbę operacji z uwzględnieniem pewnej funkcji kosztu, w której literały i operacje logiczne posiadają ściśle określone wagi. Metody najbardziej rozpowszechnione[1]:

- przekształcanie wyrażeń zgodnie z zasadami obowiązującymi w algebrze Boole'a (patrz Przykład 3.1).
- metoda siatek Karnaugh'a,
- metoda Quine'a - McCluskey'a,
- metoda Kazakowa,
- metoda tablic niezgodności,
- metoda bezpośredniego przeszukiwania.

Układ zrealizowany w oparciu o minimalną postać wyrażenia zazwyczaj zawiera minimalną liczbę elementów logicznych. Odpowiednio przeprowadzona minimalizacja pozwala określić rozwiązanie wolne od hazardu.

Tabela 4.1

kod Gray'a			
g_3	g_2	g_1	g_0
0	0	0	0
0	0	0	1
0	0	1	1
0	0	1	0
0	1	1	0
0	1	1	1
0	1	0	1
0	1	0	0
1	1	0	0
1	1	0	1
1	1	1	1
1	1	1	0
1	0	1	0
1	0	1	1
1	0	0	1
1	0	0	0

Minimalizacja metodą siatek Karnaugh'a

Algorytm określania minimalnej postaci sumy

1. Wartości funkcji Z wpisać do siatki Karnaugh'a. Wiersze siatki Karnaugh'a zakodowane są kombinacjami wartości części argumentów funkcji Z, a kolumny kombinacjami wartości pozostałej części argumentów tej funkcji. Do kodowania binarnego wierszy i kolumn używa się kodu cyklicznego zazwyczaj Gray'a. Tabela 4.1 przedstawia kolejne pozycje kodu Gray'a dla czterech bitów. Z wag argumentów przyjętych w kanonicznej postaci dziesiętnej funkcji wynika opis dziesiętny siatki.
2. Kratki zawierające wartości „1" należy połączyć w grupy wg zasad:
   - grupa powinna zawierać 2^n kratek, gdzie n jest liczbą całkowitą,
   - jeżeli grupa jest położona po obu stronach osi symetrii: siatki, połówki siatki, ćwiartki siatki, .. itd., to musi być symetryczna względem tej osi,
   - stany Φ należy wykorzystać do powiększenia grup,
   - chcąc otrzymać rozwiązanie wolne od hazardu[2] należy zapewnić, aby każda para sąsiednich logicznie stanów „1" (różniących się w opisie binarnym tylko na jednej pozycji) należała do przynajmniej jednej wspólnej grupy.

[1] Uwzględnione w programach dydaktycznych dostępnych w: Min-siat.zip, Qn-h-szk.zip, Quin-wiz.zip, Kaz-wiz.zip, Min-tn.zip, Mult-syn.zip, na witrynie internetowej: http://zmitac.iinf.polsl.gliwice.pl

[2] Zagadnienie uwzględnione w programach dostępnych w: Haz-stat.zip, H-st-win.zip Kaz-wiz.zip, Qn-h-szk.zip Quin-wiz.zip na wyżej wymienionej witrynie.

3. Grupę opisuje wyrażenie logiczne w postaci iloczynu literałów odpowiadającym tylko tym argumentom, od których zależy wartość funkcji w rozważanej grupie (czyli tych, które w opisie kratek rozważanej grupy posiadają stałą wartość). Dany argument występuje w postaci prostej, jeżeli na jego pozycji w opisie rozważanej grupy stanów występuje wartość binarna „1" lub w postaci zanegowanej, jeżeli na jego pozycji występuje wartość binarna „0".
4. Minimalna postać sumy dla Z jest sumą wyrażeń z punktu 3.

Minimalna postać iloczynu

Określa się ją algorytmem podanym dla minimalnej postaci sumy z uwzględnieniem analogii:

minimalna postaci sumy	-	minimalna postaci iloczynu
1	-	0
0	-	1
iloczyn literałów	-	suma literałów
suma wyrażeń	-	iloczyn wyrażeń

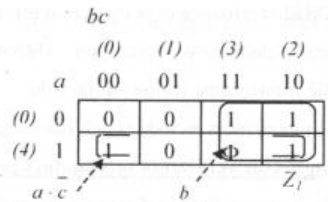
Przykład 4.1

Dana jest funkcja *nie w pełni określona* :

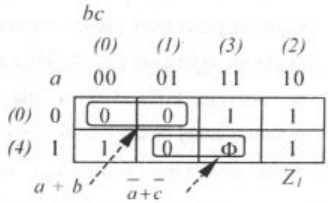
$$\begin{cases} Z_1 = \Sigma(2, 3, 4, 6)_{abc} \\ Z_1 = \Pi(0, 1, 5)_{abc} \end{cases}$$

Określenie minimalnej postaci sumy

Wartości funkcji zapisano w siatce Karnaugha (rys. 4-1). Siatka opisana jest dwójkowym kodem Gray'a. Z wag argumentów: a - 2^2, b - 2^1, c - 2^0 wynika opis dziesiętny siatki. W siatce zaznaczono dwie grupy „jedynkowe". Do powiększenia jednej z nich wykorzystano stan Φ. Minimalna postać sumy: $Z_1 = a \cdot \bar{c} + b$ jest wolna od hazardu, ponieważ każda para sąsiednich logicznie stanów „1": (2 i 6, 2 i 3, 4 i 6)$_{abc}$, należy do przynajmniej jednej wspólnej grupy.



Rys. 4-1



Rys. 4-2

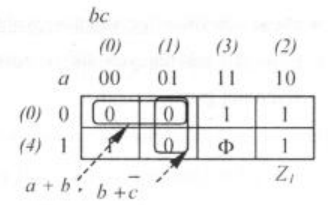
Określenie minimalnej postaci iloczynu

Dla Z_1 w siatce podanej na rys. 4-2 zaznaczono dwie grupy „zerowe". Dla ich powiększenia wykorzystano stan Φ. Z opisu tych grup wynika minimalna postać iloczynu: $Z_1 = (a + b) \cdot (\bar{a} + \bar{c})$.

Wyrażenie to zawiera hazard, bo nie zapewnia pokrycia żadną wspólną grupą pary stanów „0": (1 i 5)$_{abc}$.

Można utworzyć inne grupy „zerowe" (rys. 4-3), z których wynika postać wyrażenia: $Z_1 = (a + b) \cdot (b + \bar{c})$.

Ta postać zapewnia rozwiązanie wolne od hazardu.



Rys. 4-3

Przykład 4.2

Dana jest funkcja *w pełni określona*:

$$Z_2 = \Sigma(3, 8, 10, 11, 12, 14, 16, 17, 19, 20, 21, 24, 25, 27, 28, 29)_{abcde}$$

a)

	c d e	(0) 000	(1) 001	(3) 011	(2) 010	(6) 110	(7) 111	(5) 101	(4) 100
(0)	a b 0 0	0	0	1	0	0	0	0	0
(8)	0 1	1	0	1	1	1	0	0	1
(24)	1 1	1	1	1	0	0	0	1	1
(16)	1 0	1	1	1	0	0	0	1	1

Z_2

b)

	c d e	(0) 000	(1) 001	(3) 011	(2) 010	(6) 110	(7) 111	(5) 101	(4) 100
(0)	a b 0 0	0	0	1	0	0	0	0	0
(8)	0 1	1	0	1	1	1	0	0	1
(24)	1 1	1	1	1	0	0	0	1	1
(16)	1 0	1	1	1	0	0	0	1	1

Z_2

Rys. 4-4

Minimalna postać sumy, bez eliminacji hazardu wynika z grup zaznaczonych linią ciągłą na rys. 4-4:

$$Z_2 = \bar{c} \cdot d \cdot e + \bar{a} \cdot b \cdot \bar{e} + a \cdot \bar{d}$$

Zawiera ona hazard przy zmianach:

$(01011 \Leftrightarrow 01010)_{abcde}$ $(01000 \Leftrightarrow 11000)_{abcde}$

$(11001 \Leftrightarrow 11011)_{abcde}$ $(10001 \Leftrightarrow 10011)_{abcde}$

$(01100 \Leftrightarrow 11100)_{abcde}$

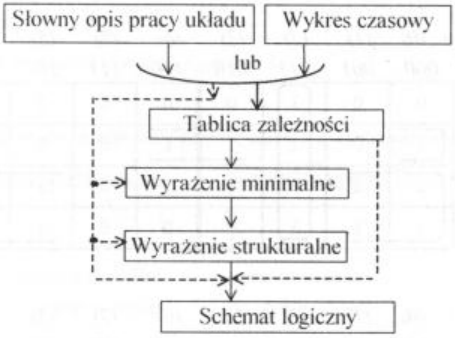
Dla eliminacji hazardu należy uzupełnić zbiór grup o grupy antyhazardowe, zaznaczone linią przerywaną na rys. 4-4b.

Uzupełnione wyrażenie wolne od hazardu ma postać:

$$Z_2 = \bar{c} \cdot d \cdot e + \bar{a} \cdot b \cdot \bar{e} + a \cdot \bar{d} + \underbrace{\bar{a} \cdot b \cdot \bar{c} \cdot d + b \cdot \bar{d} \cdot \bar{e} + a \cdot \bar{c} \cdot e}_{\text{wyrażenie antyhazardowe}}$$

5. SYNTEZA UKŁADÓW KOMBINACYJNYCH

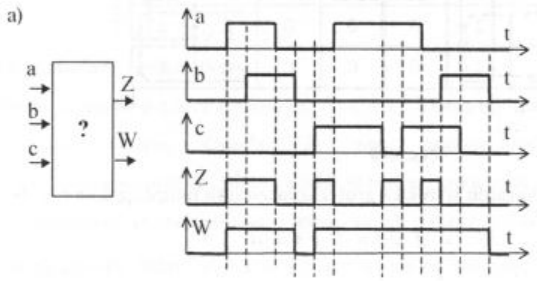
Algorytm syntezy układu kombinacyjnego przedstawia rys. 5-1.



Rys. 5-1

Przykład 5.1

Należy przeprowadzić syntezę logiczną układu opisanego wykresem czasowym (rys. 5-2a).

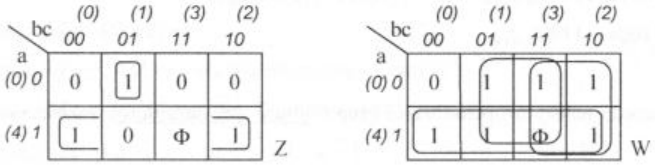


b)

a b c	Z	W
0 0 0	0	0
0 0 1	1	1
0 1 0	0	1
0 1 1	0	1
1 0 0	1	1
1 0 1	0	1
1 1 0	1	1
1 1 1	Φ	Φ

Rys. 5-2

Z wykresu czasowego wynika tablica zależności (rys. 5-2b) i siatki zależności (rys. 5-3).



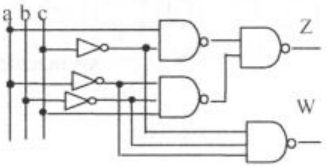
Rys. 5-3

Wyrażenia minimalne: $Z=\bar{a}\cdot\bar{b}\cdot c+a\cdot\bar{c}$; $W=a+b+c$

Wyrażenia strukturalne dla realizacji na elementach NAND:

$$Z=\overline{\overline{\bar{a}\cdot\bar{b}\cdot c}\cdot\overline{a\cdot\bar{c}}}\,; \qquad W=\overline{\bar{a}\cdot\bar{b}\cdot\bar{c}}$$

Schemat logiczny realizacji z zastosowaniem elementów NAND przedstawia rys. 5-4.



Rys. 5-4

Przykład 5.2

Rozważana jest synteza konwertera kodu Watts'a na kod pseudopierścieniowy. Układ konwertera opisany jest tablicą zależności (rys 5-5). Rozwiązanie dotyczy przypadku, kiedy w stanach nie należących do kodu Watts'a (nie ujętych w tablicy na rys. 5-5) sygnały wyjściowe są w stanie 0.

Kod Watts'a				Kod pseudopierścieniowy				
D	C	B	A	P_4	P_3	P_2	P_1	P_0
0	0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	0	1
0	0	1	1	0	0	0	1	1
0	0	1	0	0	0	1	1	1
0	1	1	0	0	1	1	1	1
1	1	1	0	1	1	1	1	1
1	0	1	0	1	1	1	1	0
1	0	1	1	1	1	1	0	0
1	0	0	1	1	1	0	0	0
1	0	0	0	1	0	0	0	0

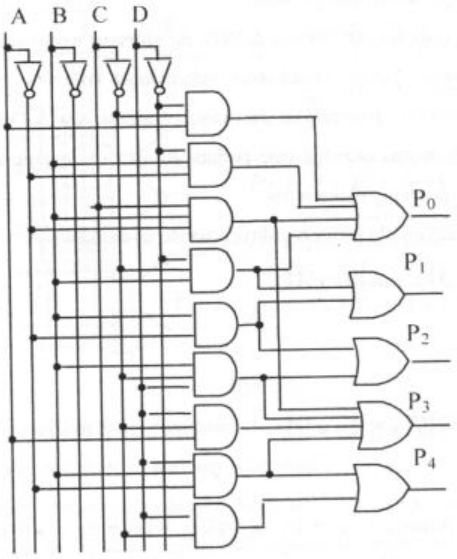
Rys. 5-5

Wolne do hazardu minimalne wyrażenia o postaci sumy dla poszczególnych wyjść są następujące:

$$P_0 = \overline{D}\,\overline{C}A + \overline{D}B\overline{A} + CB\overline{A} + \overline{D}\,\overline{C}B, \qquad P_1 = \overline{D}\,\overline{C}B + B\overline{A}, \qquad P_2 = D\overline{C}B + B\overline{A},$$

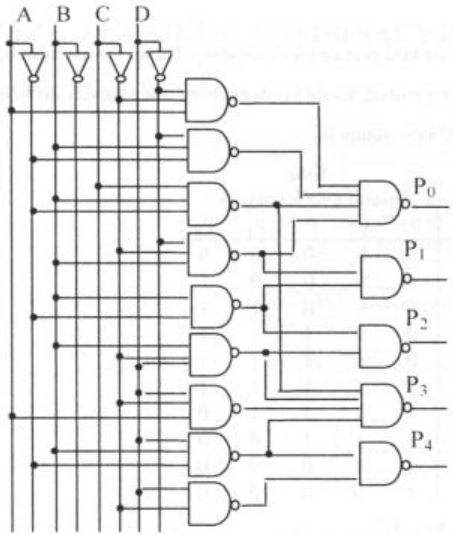
$$P_3 = D\overline{C}A + DB\overline{A} + CB\overline{A} + D\overline{C}B, \qquad P_4 = D\overline{C} + DB\overline{A}.$$

Rysunek 5-6 przedstawia schemat realizacji układu w oparciu o podstawowy system funkcjonalnie pełny.



Rys. 5-6

Wyrażenie strukturalne dla realizacji układu w oparciu o elementy NAND (rys. 5-7) otrzymuje się korzystając z prawa negacji.



$$P_0 = \overline{\overline{D\,\overline{C}\,A}\;\;\overline{D\,B\,\overline{A}}\;\;\overline{C\,B\,\overline{A}}\;\;\overline{D\,\overline{C}\,B}}.$$

$$P_1 = \overline{\overline{D\,\overline{C}\,B}\;\;\overline{B\,\overline{A}}}.$$

$$P_2 = \overline{\overline{D\,\overline{C}\,B}\;\overline{B\,\overline{A}}}.$$

$$P_3 = \overline{\overline{D\,\overline{C}\,A}\;\;\overline{D\,B\,\overline{A}}\;\;\overline{C\,B\,\overline{A}}\;\;\overline{D\,\overline{C}\,B}}.$$

$$P_4 = \overline{\overline{D\,\overline{C}}\;\;\overline{D\,B\,\overline{A}}}.$$

Rys. 5-7

Można z jego pomocą określić wyrażenia, w których występują tylko operacje negacji iloczynów literałów. W takiej strukturze występują trzy poziomy elementów.

Porównując wyrażenia o postaci sumy z odpowiadającymi im schematami realizacji w oparciu o elementy NAND, można zauważyć następującą zasadę:

Każdemu składnikowi sumy odpowiada element NAND, na który są wprowadzone sygnały odpowiadające literałom w danym składniku. Sumie składników odpowiada końcowy (wyjściowy) element NAND, na który wprowadzone są wyjścia z poprzednio omówionych elementów NAND.

W oparciu o postać sumy również można określić (nie podane tutaj) rozwiązanie z zastosowaniem elementów NOR. Będzie ono posiadało cztery poziomy elementów.

Minimalne wyrażenia o postaci iloczynu dla poszczególnych wyjść są następujące:

$$P_0 = (B + A)(\overline{D} + C)(\overline{C} + \overline{A})(\overline{C} + B)(\overline{D} + \overline{A}),$$

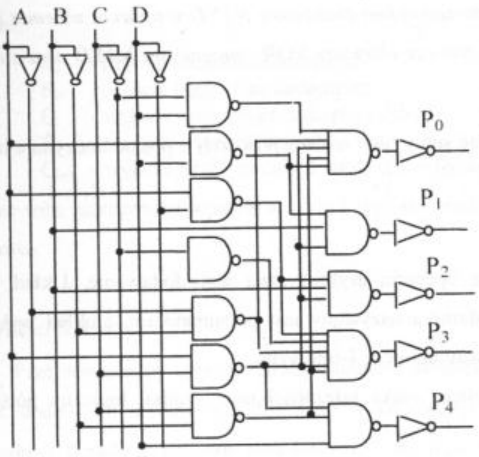
$$P_1 = B(\overline{D} + \overline{A})(\overline{C} + \overline{A}),$$

$$P_2 = B(D + \overline{A})(\overline{C} + \overline{A}),$$

$$P_3 = (B + A)(D + C)(\overline{C} + \overline{A})(\overline{C} + B)(D + \overline{A}),$$

$$P_4 = D(\overline{C} + B)(\overline{C} + \overline{A}).$$

Przekształcenie ich do postaci (rys. 5-8), w której występują tylko operacje negacji iloczynów literałów pozwala określić realizację układu na elementach NAND. W takiej strukturze występują cztery poziomy elementów.



$$P_0 = \overline{\overline{B}\,\overline{A}\;\overline{D}\overline{C}\;\overline{C}\,\overline{A}\;\overline{C}\overline{B}\;\overline{D}\overline{A}},$$

$$P_1 = \overline{B\;\overline{D}\overline{A}\;\overline{C}\overline{A}},$$

$$P_2 = \overline{B\;\overline{D}\overline{A}\;\overline{C}\overline{A}},$$

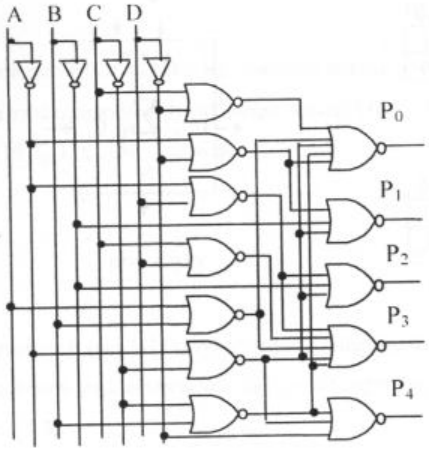
$$P_3 = \overline{\overline{B}\,\overline{A}\;\overline{D}\overline{C}\;\overline{C}\overline{A}\;\overline{C}\overline{B}\;\overline{D}\overline{A}},$$

$$P_4 = \overline{D\;\overline{C}\overline{B}\;\overline{C}\overline{A}}.$$

Rys. 5-8

W oparciu o minimalną postać iloczynu wygodniej jest realizować układ z zastosowaniem elementów NOR (rys. 5-9). W takiej strukturze występują trzy poziomy elementów.

Wyrażenie strukturalne dla realizacji w oparciu o elementy NOR otrzymuje się korzystając z prawa negacji. Można z jego pomocą określić wyrażenia, w których występują tylko operacje negacji sum literałów.



$$P_0 = \overline{\overline{B+A}+\overline{D}+C+\overline{C}+A+\overline{C}+B+\overline{D}+A},$$

$$P_1 = \overline{\overline{B}+\overline{D}+A+\overline{C}+A},$$

$$P_2 = \overline{\overline{B}+D+A+\overline{C}+A},$$

$$P_3 = \overline{\overline{B+A}+\overline{D+C}+\overline{C}+A+\overline{C}+B+\overline{D}+A},$$

$$P_4 = \overline{\overline{D}+\overline{C}+B+\overline{C}+A}.$$

Rys. 5-9

Porównując wyrażenia o postaci iloczynu z odpowiadającymi im schematami realizacji w oparciu o elementy NOR, można zauważyć zasadę podobną do podanej wcześniej.

Każdemu czynnikowi sumy odpowiada element NOR, na który są wprowadzone sygnały odpowiadające literałom w danym czynniku. Iloczynowi czynników odpowiada końcowy (wyjściowy) element NOR, na który wprowadzone są wyjścia z poprzednio omówionych elementów NOR.

Warto też zaznaczyć wnioski wynikające z porównania realizacji układu z zastosowaniem elementów NOR (rys. 5-9) i realizacji z zastosowaniem elementów NAND (rys. 5-8) w oparciu o tę samą postać np. iloczynu.

Aby otrzymać schemat realizacji układu z zastosowaniem elementów NAND w oparciu schemat realizacji układu z zastosowaniem elementów NOR, wystarczy elementy NOR zamienić na NAND oraz zanegować wejścia i wyjścia.

Ta zasada dotyczy zarówno realizacji opartej o postać sumy jak i realizacji opartej o postać iloczynu a także odwrotnej zamiany (elementów NAND na NOR).

Przykład 5.3

Jak pokazano w rozdz.1 najbardziej podstawową operacją arytmetyczną jest dodawanie. Układ cyfrowy wykonujący dodawanie dwóch cyfr dodajnej i dodajnika nazywany jest półsumatorem, a układ wykonujący dodawanie dwóch cyfr i przeniesienia nazywamy jest sumatorem 1-bitowym.

Na rys. 5-10 przedstawiono kolejno tablice, siatki zależności oraz symbol logiczny półsumatora (rys. 5-10a) i sumatora 1-bitowego (rys. 5-10b).

a)

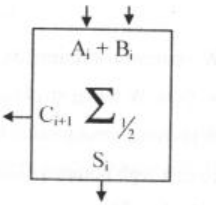
A_i	B_i	S_i	C_{i+1}
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

S_i

A_i \ B_i	0	1
0	0	1
1	1	0

C_{i+1}

A_i \ B_i	0	1
0	0	0
1	0	1



b)

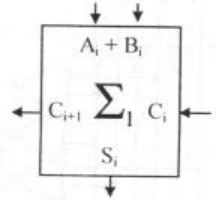
A_i	B_i	C_i	S_i	C_{i+1}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

S_i

A_i \ B_iC_i	00	01	11	10
0	0	1	0	1
1	1	0	1	0

C_{i+1}

A_i \ B_iC_i	00	01	11	10
0	0	0	1	0
1	0	1	1	1



Rys. 5-10

Z siatek zależności uzyskuje się odpowiednio funkcje wyjść:

- dla półsumatora $S_i = \overline{A}_i B_i + A_i \overline{B}_i = A_i \oplus B_i$

  $C_{i+1} = A_i B_i$

- dla sumatora 1-bitowego:

$$S_i = \overline{A}_i \overline{B}_i C_i + A_i B_i C_i + \overline{A}_i B_i \overline{C}_i + A_i \overline{B}_i \overline{C}_i = (\overline{A}_i \overline{B}_i + A_i B_i) C_i + (\overline{A}_i B_i + A_i \overline{B}_i) \overline{C}_i =$$

$$= \overline{A_i \oplus B_i}\, C_i + (A_i \oplus B_i) \overline{C}_i = A_i \oplus B_i \oplus C_i$$

$$C_i = A_i B_i + A_i C_i + B_i C_i$$

przy czym:

A_i -wejście i-tego bitu dodajnej;
B_i -wejście i-tego bitu dodajnika;
C_i - wejście przeniesienia do pozycjii-tej;
S_i - wyjście i-tego bitu sumy;
C_{i+1} - wyjście przeniesienia do pozycji (i+1)-szej.

Łącząc ze sobą szeregowo (poprzez wejścia i wyjścia przeniesienia) sumatory 1-bitowe uzyskuje się sumatory wielobitowe.

Przykład 5.4

Przeprowadzimy syntezę tetrady sumatora dziesiętnego pracującego w kodzie 8421 pozwalającej wykonywać dodawanie dwóch cyfr dziesiętnych kodowanych dwójkowo (${}^{X}T_i$, ${}^{Y}T_i$) i przeniesienia (P_i) z poprzedniej tetrady (sumator tetradowy). Przy jego realizacji układowej należy uwzględnić specyfikę wykonywania dodawania pojedynczych cyfr liczb dziesiętnych kodowanych dwójkowo (porównaj podrozdział 1.3). W przypadku liczb dziesiętnych o cyfrach przedstawionych w kodzie 8421 korekcja polega na dodaniu w tetradziewartości $(0110)_2 = (6)_{10}$, o ile uzyskana wartość sumy z dodawania odpowiedników dwójkowych cyfr wynosi:

$$10 \leq T' \leq 19$$

Przeniesienie P_{i+1} do bardziej znaczącej tetrady powinno być wytworzone, gdy istnieje przeniesienie C_4 z najbardziej znaczącego bitu dwójkowego tetrady ($16 \leq T' \leq 19$) lub uzyskana suma jest większa od 9, a mniejsza od 16 ($10 \leq T' \leq 15$). Tak więc:

$$P_{i+1} = C_4 + C'$$

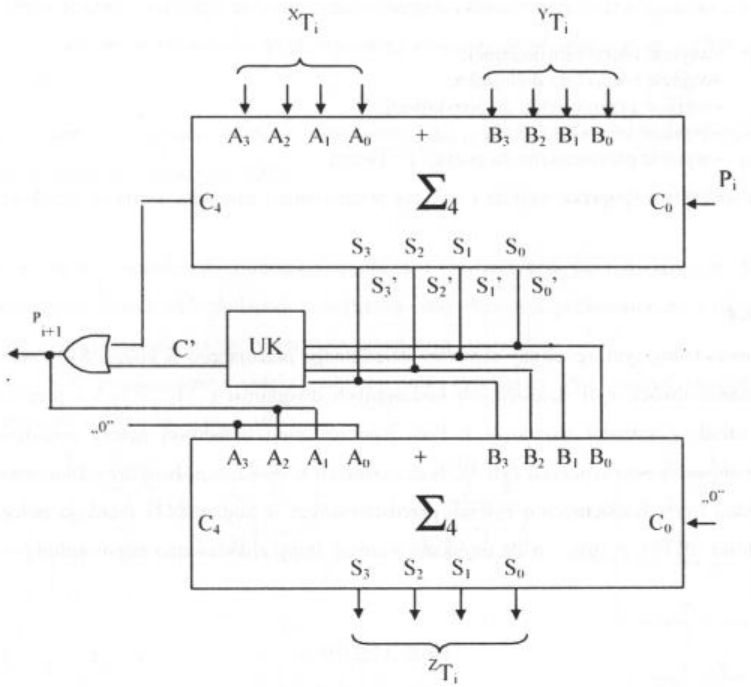
przy czym $C' = 1$ gdy $10 \leq T' \leq 15$.

Do realizacji (rys.5-11) wykorzystano sumatory 4-bitowe. Struktura układu kombinacyjnego UK określona jest wyrażeniem logicznym odpowiadającym sygnałowi C', który można określić z siatki zależności przedstawionej na rys.5-12.

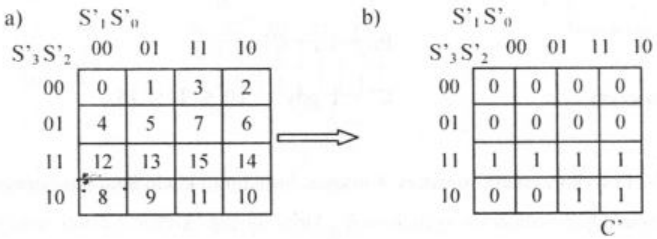
$$C' = S_3' S_2' + S_3' S_1'$$

Ostatecznie więc sygnał przeniesienia

$$P_{i+1} = C_4 + S_3' S_2' + S_3' S_1'$$

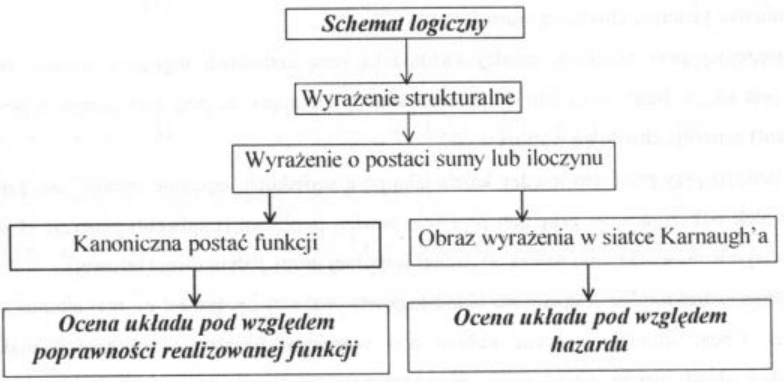


Rys. 5-11



Rys. 5-12

6. DYNAMIKA UKŁADÓW KOMBINACYJNYCH



Rys. 6-1

Algorytm analizy układu kombinacyjnego przedstawia rysunek 6-1. Wyrażenie strukturalne opisujące układ może być różne od postaci sumy i różne od postaci iloczynu. Można podać ścisły algorytm analizy ze względu na hazard dla układu opisanego wyrażeniem o postaci sumy lub iloczynu.

Posługując się obrazem wyrażenia w siatce Karnaugh'a można w łatwy sposób przeprowadzić analizę wyrażenia pod względem hazardu.

Obraz wyrażenia w siatce Karnaugh'a

W siatce Karnaugha'a tworzony jest obraz dla wyrażenia `Wk` o postaci alternatywnej (koniunkcyjnej) według następujących zasad:

- kratki o wartości `1` odpowiadają stanom `ek` funkcji opisanej wyrażeniem `Wk`, a kratki o wartości `0` odpowiadają stanom `ec` tej funkcji.
- implikantowi (implicentowi), który nie jest tożsamościowo równy `0` (`1`) odpowiada grupa kratek jedynkowych (zerowych) zaznaczonych obwódką,
- implikantowi (implicentowi) tożsamościowo równemu `0` (`1`) odpowiadają pary stanów zaznaczone parami strzałek i wskazujące przejścia, przy których ten implikant (implicent) może generować chwilową wartości `1` (`0`).

Ocena wyrażenia pod względem hazardu w oparciu o jego obraz w siatce Karnaugh'a

Obraz wyrażenia alternatywnego (koniunkcyjnego) w siatce świadczy, że w wyrażeniu występuje:

- `HSd` (`HSn`) przy przejściu między każdą taką parą sąsiednich logicznie stanów `ek` (`ec`), że kratki dotyczące tej pary stanów nie należą do żadnej wspólnej grupy jedynkowej (zerowej), czyli nie znajdują się we wspólnej obwódce.

- HSn (HSd) przy przejściu między każdą taką parą sąsiednich logicznie ec (ek), że istnieje para strzałek wskazująca przejście, przy którym pewien implikant (implicent) tożsamościowo równy 0 (1) wyrażenia Wk generuje chwilową wartość 1(0).
- HD występujący przy przejściu między każdą taką parą sąsiednich logicznie stanów, że jeden z tych stanów jest ek a drugi ec i istnieje para strzałek wskazująca, że przy tym przejściu pewien implikant (implicent) generuje chwilową wartość 1(0).
- wHSd (wHSn) przy przejściu między każdą taką parą sąsiednich logicznie stanów ek (ec), że istnieje para strzałek wskazująca, że przy tym przejściu pewien implikant (implicent) generuje chwilową wartość 1(0), a wymienione stany nie należą do żadnej wspólnej grupy jedynkowej (zerowej).

Wyrażenie strukturalne W opisujące układ logiczny może być różne od postaci alternatywnej i postaci koniunkcyjnej. Chcąc umożliwić ocenę układu pod względem hazardu w oparciu o analizę wyrażenia opisującego ten układ, trzeba podać ścisły algorytm przekształcania wyrażenia strukturalnego do postaci alternatywnej lub koniunkcyjnej z uwzględnieniem hazardu.

Proces analizy pod względem hazardu można więc sprowadzić do dwóch etapów:

ETAP I obejmujący przekształcenie wyrażenia strukturalnego W do wyrażenia Wk o postaci alternatywnej lub koniunkcyjnej, równoważnej logicznie i pod względem hazardu,

ETAP II obejmujący analizę pod względem hazardu dla wyrażenia Wk z ETAPU I.

Dwupoziomowe układy NAND lub NOR o wejściach prostych i zanegowanych opisują wyrażenia, które już po zastosowaniu prawa negacji przyjmują łatwą do analizy normalną postać alternatywną lub normalną postać koniunkcyjną. Normalna postać alternatywna (koniunkcyjna) jest wolna od HSn i HD (HSd i HD) i może zawierać jedynie HSd (HSn).

Znacznie trudniejszy do analizy jest układ opisany wyrażeniem różnym od normalnego. Na przykład trzy-poziomowy układ NAND lub trzy-poziomowy układ NOR, który na każdym poziomie może posiadać wielowejściowe elementy, opisany jest wyrażeniem strukturalnym, które po zastosowaniu prawa negacji przyjmuje postać Wn mieszaną, różną od postaci alternatywnej i koniunkcyjnej.

Dla zachowania równoważności nie tylko logicznej ale też pod względem hazardu, podczas przekształcania nie można korzystać z tych zależności, których lewa i prawa strona są różne pod względem hazardu, chociażby one były równe logicznie.

Nie przestrzegając tych zasad, nie można ocenić poprawnie rozwiązania układu pod względem hazardu w oparciu o analizę wyrażenia przekształconego. Wtedy nawet łączna ocena wyrażeń obu postaci (alternatywnej i koniunkcyjnej) nie zapewnia pełnej oceny rozwiązania.

Przykłady zależności różnych pod względem hazardu chociaż prawdziwych logicznie

$$x_i \cdot \overline{x_i} = 0 \quad (1) \qquad x_i + \overline{x_i} = 1 \quad (2)$$

$$x_i\left(x_i + \overline{x_i}\right) = x_i \quad (3) \qquad x_i + x_i \cdot \overline{x_i} = x_i \quad (4)$$

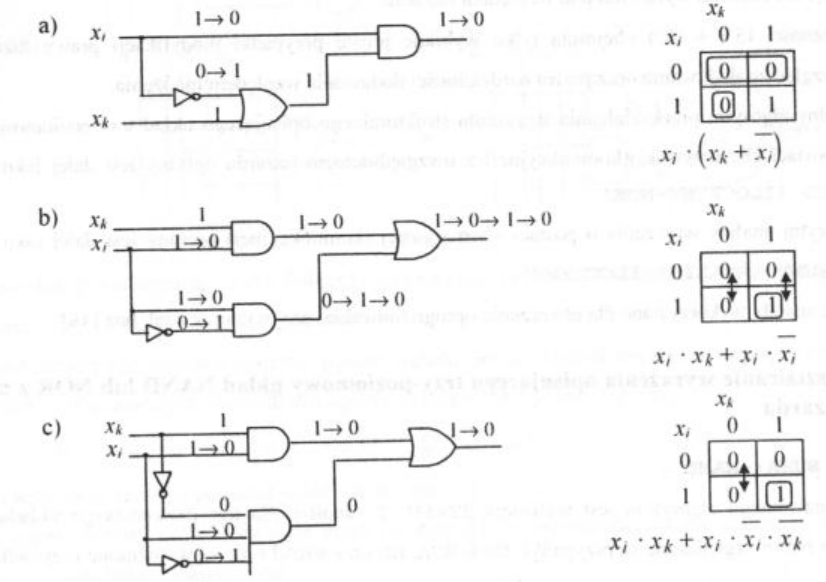
$$x_i \cdot \left(x_k + \overline{x_i}\right) = x_i \cdot x_k + x_i \cdot \overline{x_i} \quad (5) \qquad x_i + x_k \cdot \overline{x_i} = \left(x_i + x_k\right) \cdot \left(x_i + \overline{x_i}\right) \quad (6)$$

$$\left(x_i + x_k\right) \cdot \left(x_l + \overline{x_i}\right) = x_i \cdot x_l + x_k \cdot x_l + x_i \cdot \overline{x_i} + x_k \cdot \overline{x_i} \quad (7)$$

$$x_i \cdot x_k + x_l \cdot \overline{x_i} = \left(x_i + x_l\right) \cdot \left(x_k + x_l\right) \cdot \left(x_i + \overline{x_i}\right) \cdot \left(x_k + \overline{x_l}\right) \quad (8)$$

Wyrażenia: $x_i \cdot \overline{x_i}$ oraz $x_i + \overline{x_i}$ występujące w lewych stronach zależności (1) ÷ (4) są hazardogenne, podczas gdy prawe strony tych zależności są wolne od hazardu.

Zależności (5) ÷ (8) dotyczą stosowania praw rozdzielności: mnożenia względem dodawania oraz rozdzielności dodawania względem mnożenia w przypadku, gdy w wyniku ich stosowania otrzymuje się odpowiednio implikant tożsamościowo równy 0 lub implicent tożsamościowo równy 1. Przykładowo różnice pod względem hazardu dla układów realizujących lewą i prawą stronę zależności (5) ilustruje obraz wyrażeń w siatkach na rysunku 6-2a i 6-2b.



Rys. 6-2

Układ na rys. 6-2a posiada mniej przypadków hazardu niż układ na rys. 6-2b. Widać to zarówno z analizy zmian sygnałów na schematach logicznych jak i z porównania obrazów wyrażeń opisujących te układy w siatkach Karnaugh'a. Obydwa wymienione układy posiadają HSn przy przejściu $(00 \Leftrightarrow 10)_{x_i x_k}$.

Ponadto układ z rysunku 6-2b posiada HD przy przejściu: $(01 \Leftrightarrow 11)_{x_i x_k}$, którego nie posiada układ z rys. 6-2a. Układ z rys. 6-2c jest równoważny układowi z rys. 6-2a logicznie oraz pod względem hazardu.

Zatem przy stosowaniu prawa rozdzielności mnożenia względem dodawania dla lewej strony zależności (5), celem otrzymania wyrażenia równoważnego nie tylko logicznie ale również pod względem hazardu, implikant hazardogenny $x_i \cdot \overline{x_i}$ występujący w prawej stronie zależności (5) należy zmodyfikować do postaci $x_i \cdot \overline{x_i} \cdot \overline{x_k}$ (rys. 6-2c).

Z podobnej analizy pozostałych przypadków wynika, że dla uwzględnienia zagadnień hazardu potrzebna jest modyfikacja zależności (5) ÷ (8) do postaci (5') ÷ (8').

Zależności prawdziwe nie tylko logicznie ale również pod względem hazardu

$$x_i \cdot (x_k + \overline{x_i}) = x_i \cdot x_k + x_i \cdot \overline{x_i} \cdot \overline{x_k} \qquad (5')$$

$$x_i + x_k \cdot \overline{x_i} = (x_i + x_k) \cdot (x_i + \overline{x_i} + \overline{x_k}) \qquad (6')$$

$$(x_i + x_k) \cdot (x_l + \overline{x_i}) = x_i \cdot x_l + x_k \cdot x_l + x_i \cdot \overline{x_i} \cdot \overline{x_k} \cdot \overline{x_l} + x_k \cdot \overline{x_i} \qquad (7')$$

$$x_i \cdot x_k + x_l \cdot \overline{x_i} = (x_i + x_l) \cdot (x_k + x_l) \cdot (x_i + \overline{x_l} + \overline{x_k} + \overline{x_l}) \cdot (x_k + \overline{x_i}) \qquad (8')$$

Prawo negacji nie zmienia wyrażenia pod względem hazardu.

Zależności (5') ÷ (8') obejmują tylko wybrane proste przypadki modyfikacji prawa rozdzielności mnożenia względem dodawania oraz prawa rozdzielności dodawania względem mnożenia.

Ogólny algorytm przekształcania wyrażenia strukturalnego opisującego układ trzy-poziomowy NAND (NOR) do postaci alternatywnej (koniunkcyjnej) z uwzględnieniem hazardu opisany jest dalej jako algorytm SUMA<-NAND (ILOCZYN<-NOR).

Ogólny algorytm analizy wyrażenia o postaci alternatywnej (koniunkcyjnej) opisany jest dalej jako algorytm ANALIZA_SUMY (ANALIZA_ILOCZYNU).

Algorytmy te zostały wykorzystane dla utworzenia oprogramowania: prz_wyr oraz anal_haz [16].

6.1. Przekształcanie wyrażenia opisującego trzy-poziomowy układ NAND lub NOR z uwzględnieniem hazardu

Algorytm SUMA<-NAND[1]

Zadaniem omawianego algorytmu jest realizacja ETAPU I analizy dla trzy-poziomowego układu NAND. W algorytmie tym uwzględnione są przypadki: HSd, HSn, HD oraz wHSd i nie są wyróżnione przypadki: wHSn ani wHD.

[1] Oparte o ten algorytm komputerowe przekształcanie wyrażeń z wizualizacją umożliwia program dydaktyczny Prz_wyr na witrynie internetowej: http://zmitac.iinf.polsl.gliwice.pl

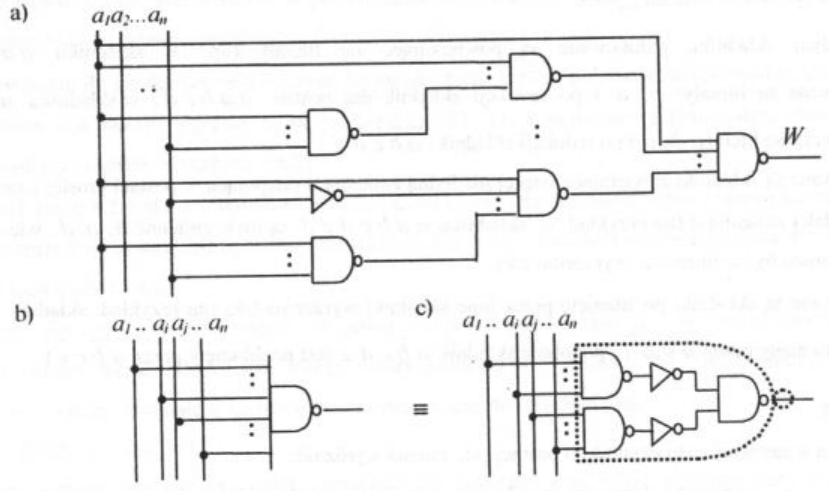
Opis algorytmu SUMA<-NAND został przedstawiony na rys.: 6-4 do 6-7 w postaci schematu blokowego.

Trzy-poziomowy układ NAND opisuje wyrażenie strukturalne W określone zapisem:

$$W = a_1 \cdot a_1 \cdot \ldots \cdot a_d \cdot \overline{a_j} \cdot \ldots \cdot \overline{a_n} \cdot \overline{a_1} \cdot \ldots \cdot \overline{a_k} \cdot \ldots \cdot \overline{a_l} \cdot \ldots \cdot \overline{a_n} \cdot \ldots \cdot \overline{a_1} \cdot \ldots \cdot \overline{a_d} \cdot \overline{a_i} \cdot \ldots \cdot \overline{a_k} \cdot \overline{a_l} \cdot \ldots \cdot \overline{a_k} \cdot \ldots \cdot \overline{a_c} \cdot \ldots \cdot \overline{a_z} \qquad (9)$$

Ponieważ układy z rys. 6-3b i 6-3c są równoważne logicznie i pod względem hazardu, to w rozważanym algorytmie przez strukturę trzy-poziomową NAND można rozumieć nie tylko układ o schemacie np. z rysunku 6-3a, ale także równoważny mu układ logiczny, w którym „więcej-wejściowe" elementy NAND są zastąpione układem opartym o „mniej-wejściowe" elementy NAND, tak jak to podają rys.: 6-3b i 6-3c.

Uwaga. Wyrażenie (9) nie odpowiada ściśle schematowi z rysunku 6-3a.



Rys. 6-3

Po zastosowaniu prawa negacji (na rys. 6-4 przekształcenie_Wn<-W) wyrażenie (9) przyjmuje postać:

$$Wn = s_1 + \ldots + s_i + \ldots + s_n \qquad (10)$$

Gdy sygnał dociera bezpośrednio na trzeci poziom układu, wtedy składnik ma postać negacji odpowiadającego mu argumentu. W pozostałych przypadkach ogólnie składnik s ma postać iloczynu:

$$s = a_1 \cdot \ldots \cdot a_d \cdot \overline{a_j} \cdot \ldots \cdot \overline{a_n} \cdot \left(\overline{a_1} + \ldots + \overline{a_k}\right) \cdot \ldots \cdot \left(\overline{a_l} + \ldots + \overline{a_n}\right) \qquad (11)$$

Czynniki tego iloczynu mogą posiadać jedną z trzech postaci:

(a) gdy sygnał dociera bezpośrednio na drugi poziom układu, odpowiadający mu argument występuje bez negacji w składniku s jako czynnik,

(b) gdy sygnał dociera na drugi poziom układu poprzez element NOT umieszczony na pierwszym poziomie, odpowiadający mu argument zanegowany występuje w składniku s jako czynnik,

(c) gdy sygnały sterują bezpośrednio elementem NAND na pierwszym poziomie, suma osobno zanegowanych odpowiadających tym sygnałom argumentów (w nawiasie), występuje w składniku s jako czynnik.

Wyrażenie ogólne Wn ma postać mieszaną (różną od postaci alternatywnej i od postaci koniunkcyjnej). Celem przekształcenia go do postaci alternatywnej należy wyrażenie każdego ze składników s, które zawierają przynajmniej jeden czynnik o postaci (c), przekształcić do postaci alternatywnej w wyniku procedury MNOŻENIE.

W wyrażeniu Wk przechowywany jest wynik przekształcenia wyrażenia Wn do postaci alternatywnej. Do Wk dopisywany jest (dodawany jest logicznie) odpowiednio:

- wynik Wm procedury MNOŻENIE, gdy s zawiera przynajmniej jeden czynnik o postaci (c)
- lub składnik s, gdy nie zawiera on czynników o postaci (c).

Dla ułatwienia analizy w ETAPIE II wyrażenie Wk poddawane jest na końcu ETAPU I następującej redukcji (na rys. 6-4 zredukuj_Wk):

- w każdym składniku redukowane są powtarzające się literały (np.: w składniku $a\,\bar{a}\,b\,c\,\bar{d}\,b\,\bar{a}$ redukowane są literały: b, $\bar{a}$ i po redukcji składnik ma postać: $a\,\bar{a}\,b\,c\,\bar{d}$: w składniku $\bar{a}\,\bar{b}\,c\,\bar{d}\,c\,\bar{b}$ redukowane są literały: $\bar{b}$, c i po redukcji składnik: $\bar{a}\,\bar{b}\,c\,\bar{d}$),
- redukowane są składniki zawierające więcej niż jedną zmienną występującą w postaci prostej i zanegowanej, zwaną dalej zmienną H (na przykład: w składniku $a\,\bar{a}\,b\,c\,d\,e\,\bar{d}$ są dwie zmienne H: a, d, więc ten składnik powinien być usunięty z wyrażenia Wk),
- redukowane są składniki pochłonięte przez inne składniki wyrażenia Wk, (na przykład: składnik $a\,\bar{a}\,b\,c\,\bar{d}$ jest pochłonięty przez $a\,\bar{a}\,b\,\bar{d}$, podobnie składnik $a\,\bar{b}\,c\,d\,\bar{e}$ jest pochłonięty przez $a\,\bar{b}\,c\,\bar{e}$).

MNOZENIE

W składniku s zawierającym czynniki o postaci (c), można wyróżnić:

- część cz1, którą stanowi iloczyn czynników o postaci (a) lub (b),
- część cz2, którą stanowi iloczyn czynników o postaci (c).

Rozważając optymalne rozwiązania trzypoziomowych struktur NAND, przypadek, gdy w cz1 występuje zmienna hazardonenna H, praktycznie nie powinien wystąpić. Dla zapewnienia kompletności algorytmu, ten przypadek jednak uwzględniono i w opisie procedury MNOZENIE (rys. 6-5) został on wyróżniony linią przerywaną. W omawianym przypadku postępowanie jest następujące:

- gdy w części cz1 występuje dokładnie jedna zmienna H, wtedy iloczyn części cz1 i cz2 przekształca się do postaci alternatywnej w wyniku stosowania prawa rozdzielności mnożenia względem dodawania bez modyfikacji, zwane dalej jako MNOŻENIE_ZWYKŁE.
- gdy w cz1 występuje więcej niż jedna zmienna H, to (uwzględniając tylko pojedyncze zmiany wejść) składnik s należy pominąć w Wk.

Gdy w cz1 nie występuje zmienna H (prosta i zanegowana), postępowanie zależy od tego, czy w cz1 występuje pewna zmienna w postaci prostej, która w postaci przeciwnej występuje w części cz2:

- jeżeli tak, to należy zastosować MNOŻENIE_ZMODYFIKOWANE,

- w przeciwnym przypadku należy zastosować MNOŻENIE_ZWYKŁE.

MNOŻENIE_ZMODYFIKOWANE

Kolejność „wymnażania" czynników składnika s ma wpływ na złożoność algorytmu przekształcenia tego składnika s do postaci alternatywnej w przypadku, gdy cz2 zawiera więcej niż jeden czynnik o postaci (c), czyli cz2 posiada postać koniunkcyjną. W opisanym w pracy algorytmie, dla jego uproszczenia wykorzystano to, że czynniki o postaci (c) zawierają tylko argumenty zanegowane.

Zatem dla przekształcenia części cz2 o postaci koniunkcyjnej do postaci alternatywnej cz2s wystarczy zastosować MNOŻENIE_ZWYKŁE.

Dla uproszczenia przebiegu dalszej części algorytmu, alternatywna postać cz2s poddawana jest redukcji do postaci cz2r. Redukowane są powtarzające się literały w składnikach oraz składniki pochłonięte przez inne składniki wyrażenia cz2s.

W dążeniu do przekształcenia iloczynu wyrażeń: cz1 · cz2r do postaci alternatywnej, dla kolejnych par p tworzony jest iloczyn logiczny I (na rys. 6-6 określ_I). Kolejną parę p tworzą dwa elementy: część cz1 oraz kolejny składnik wyrażenia cz2r.

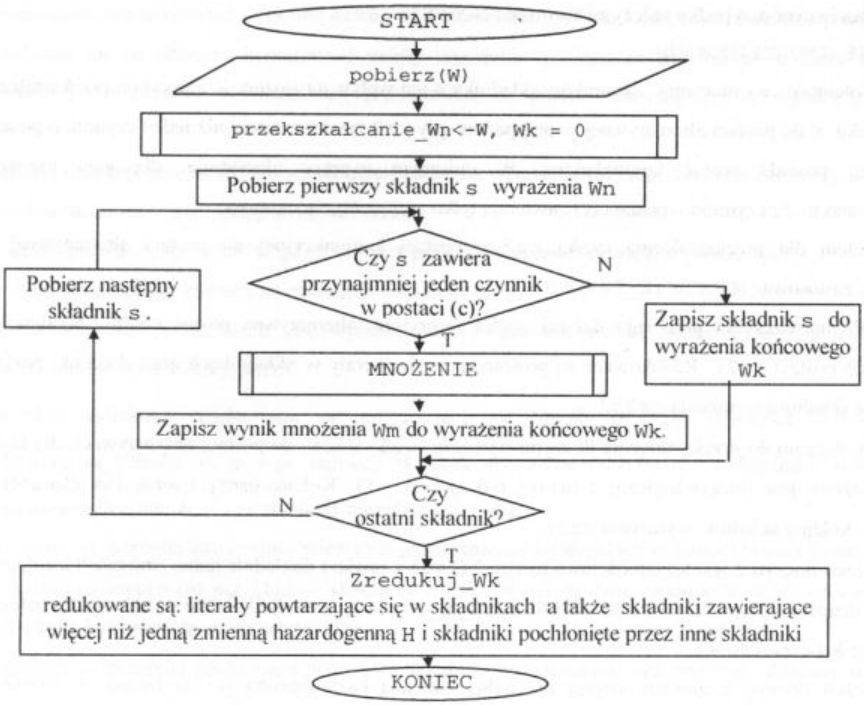
Jeżeli iloczyn I jest tożsamościowo równy 0 (I≡0) i zawiera dokładnie jedną zmienną hazardogenną H, należy I uzupełnić do postaci IU (UZUPEŁNIENIE_I_DO_IU). Zmodyfikowane wyrażenia IU należy zapisać (dodać logicznie) do Wm.

Jeżeli iloczyn I zawiera więcej niż jedną zmienną hazardogenną H, to (mając na uwadze tylko pojedyncze zmiany wejść układu) ten iloczyn należy pominąć w wyrażeniu Wm. Jeżeli iloczyn I nie jest tożsamościowo równy 0, to należy go bez uzupełniania zapisać do wyrażenia Wm.

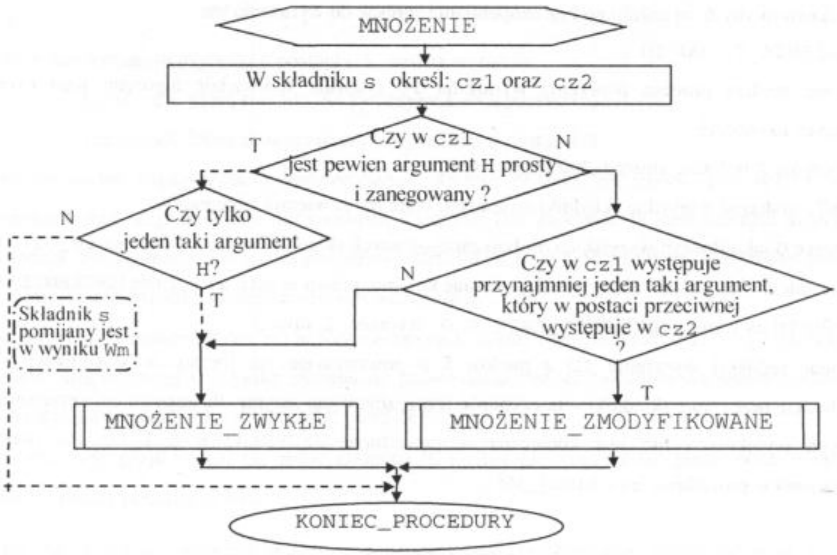
UZUPEŁNIENIE_I_ DO_IU

Szczegółowa analiza procesu tworzenia wyrażenia IU pozwala sprowadzić algorytm jego tworzenia do następujących czynności:

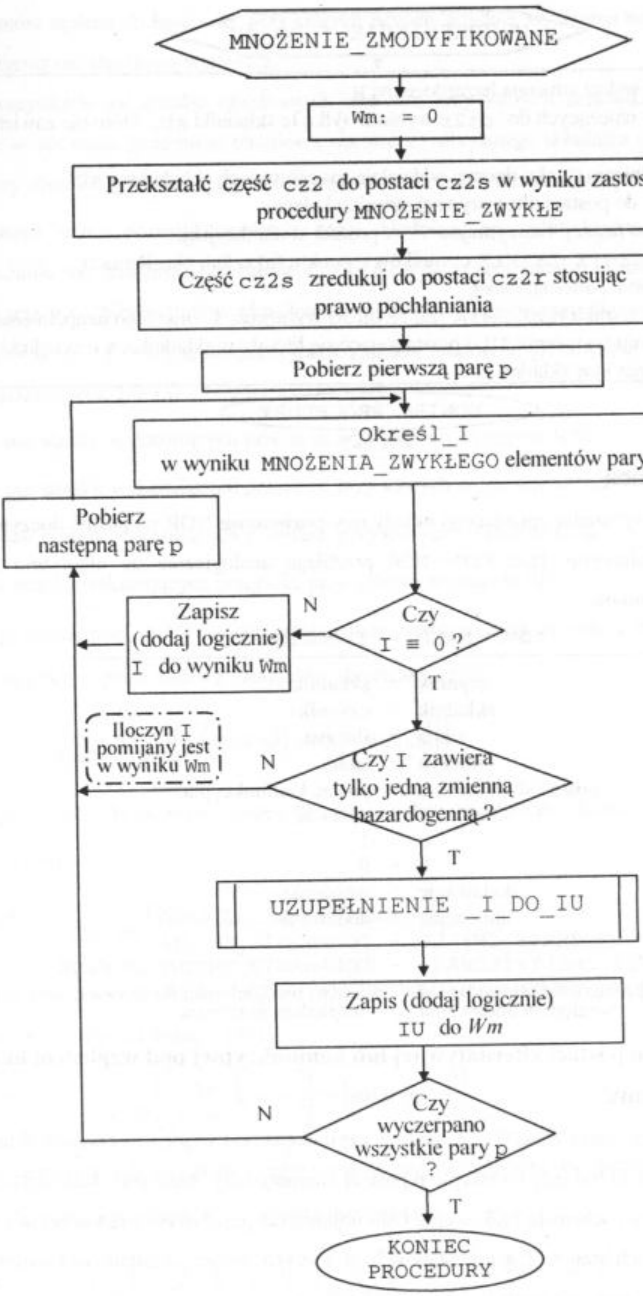
1. W iloczynie I wskazać zmienną hazardogenną H.
2. W cz2r wskazać wszystkie składniki zwane sU, które nie zawierają zmiennej H.
3. Utworzyć U jako iloczyn wszystkich osobno zanegowanych składników sU (tworzenie_U).
4. Przekształcić U do postaci alternatywnej stosując kolejno: prawo negacji a następnie MNOZENIE_ZWYKLE.
5. Określić IU w wyniku MNOZENIA_ZWYKLEGO wyrażeń: I oraz U.
6. Dokonać redukcji wyrażenia IU z punktu 5 o powtarzające się literały w składnikach i składniki pochłonięte przez inne składniki – ta czynność jest realizowana jedynie dla zapewnienia przejrzystości uzyskanych wyników i nie jest konieczna w procedurze UZUPEŁNIENIE_I_DO_IU, ponieważ jest powtarzana w procedurze zredukuj_Wk.


START
pobierz(W)
przekszkałcanie_Wn<-W, Wk = 0
Pobierz pierwszy składnik s wyrażenia Wn
Czy s zawiera przynajmniej jeden czynnik w postaci (c)?
N
T
Pobierz następny składnik s.
Zapisz składnik s do wyrażenia końcowego Wk
MNOŻENIE
Zapisz wynik mnożenia Wm do wyrażenia końcowego Wk.
Czy ostatni składnik?
N
T
Zredukuj_Wk
redukowane są: literały powtarzające się w składnikach a także składniki zawierające więcej niż jedną zmienną hazardogenną H i składniki pochłonięte przez inne składniki
KONIEC

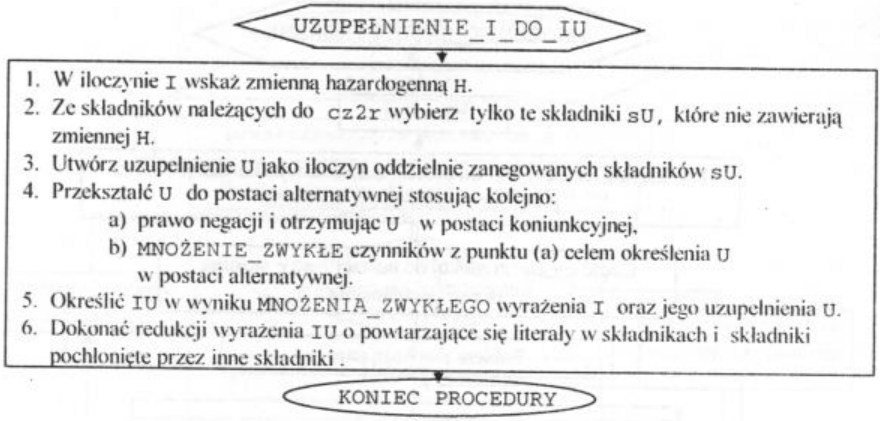
Rys. 6-4


MNOŻENIE
W składniku s określ: cz1 oraz cz2
Czy w cz1 jest pewien argument H prosty i zanegowany ?
T
N
Czy tylko jeden taki argument H?
N
T
Składnik s pomijany jest w wyniku Wm
Czy w cz1 występuje przynajmniej jeden taki argument, który w postaci przeciwnej występuje w cz2 ?
N
T
MNOŻENIE_ZWYKŁE
MNOŻENIE_ZMODYFIKOWANE
KONIEC_PROCEDURY

Rys. 6-5


MNOŻENIE_ZMODYFIKOWANE
Wm: = 0
Przekształć część cz2 do postaci cz2s w wyniku zastosowania procedury MNOŻENIE_ZWYKŁE
Część cz2s zredukuj do postaci cz2r stosując prawo pochłaniania
Pobierz pierwszą parę p
Określ_I w wyniku MNOŻENIA_ZWYKŁEGO elementów pary p
Pobierz następną parę p
Zapisz (dodaj logicznie) I do wyniku Wm
N
Czy I ≡ 0 ?
T
Iloczyn I pomijany jest w wyniku Wm
N
Czy I zawiera tylko jedną zmienną hazardogenną ?
T
UZUPEŁNIENIE _I_DO_IU
Zapis (dodaj logicznie) IU do Wm
N
Czy wyczerpano wszystkie pary p ?
T
KONIEC PROCEDURY

Rys. 6-6



Rys. 6-7

Algorytm ILOCZYN<-NOR

Algorytm przekształcania wyrażenia opisującego układy trzy-poziomowe NOR na postać iloczynu (koniunkcyjną) zwany dalej jako algorytm ILOCZYN<-NOR przebiega analogicznie do algorytmu SUMA<-NAND po zastosowaniu odpowiedników:

SUMA<-NAND		ILOCZYN<-NOR
czynnik	-	składnik
składnik	-	czynnik
suma	-	iloczyn
iloczyn	-	suma
postać alternatywna	-	postać koniunkcyjna
iloczyn I	-	suma S
0	-	1
1	-	0
dodawanie	-	mnożenie
mnożenie	-	dodawanie
MNOŻENIE_ZWYKŁE	-	DODAWANIE_ZWYKŁE
MNOŻENIE_ZMODYFIKOWANE	-	DODAWANIE_ZMODYFIKOWANE
prawo rozdzielności mnożenia względem dodawania	-	prawo rozdzielności dodawania względem mnożenia

6.2. Analiza wyrażenia o postaci alternatywnej lub koniunkcyjnej pod względem hazardu

Algorytm ANALIZA_SUMY[2]

Zadaniem algorytmu jest realizacja ETAPU II analizy, czyli ocena pod względem hazardu układu równoważnego logicznie i pod względem hazardu wyrażeniu o postaci alternatywnej. Algorytm ANALIZA_SUMY przedstawiono na rys. 6-8 w postaci schematu blokowego. Oto objaśnienia pojęć używanych w opisie.

wd - zbiór tych wszystkich stanów („grupa stanów"), w których pewien składnik wyrażenia Wk, tożsamościowo różny od 0, generuje wartość 1.

Wd - zbiór wszystkich „grup stanów" określonych jako wd.

wh - para stanów sąsiednich logicznie, przy których pewien składnik wyrażenia Wk tożsamościowo równy 0 może generować chwilową wartość 1.

Wh - zbiór wszystkich par stanów określonych jako wh, przy których składniki tożsamościowo równe 0 zawarte w Wk mogą generować chwilową wartość 1; dla danego składnika tożsamościowo równego 0, są to pary stanów sąsiednie logicznie ze względu na zmienną hazardogenną H zawarte w zbiorze stanów określonych przez pozostałe zmienne tego składnika, np.: dla składnika $a \cdot b \cdot \bar{b} \cdot d$ zawartego w wyrażeniu Wk zależnym od argumentów: a,b,c,d są to pary stanów sąsiednich logicznie ze względu na zmienną b w zbiorze stanów określonych przez pozostałe zmienne występujące w składniku, czyli: $a = 1$, $c = \Phi$, $d = 1$. Pary te można wyrazić zapisem binarnym: $\{(1001,1101), (1011,1111)\}_{abcd}$, lub zapisem dziesiętnym: $\{(9,13), (11,15)\}_{abcd}$.

Z_{HSd} - zbiór par stanów wskazujących przejścia, przy których występuje HSd.

Z_{wHSd} - zbiór par stanów wskazujących przejścia, przy których występuje wHSd.

Z_{HSn} - zbiór par stanów wskazujących przejścia, przy których występuje HSn.

Z_{HD} - zbiór par stanów wskazujących przejścia, przy których występuje HD.

Do zbioru Z_{HSd} należą pary stanów odpowiadające tym elementom qk ze zbioru stanów sąsiednich logicznie Qk', które nie są pokryte przez żaden z elementów zbioru Wd, czyli:

$$Z_{HSd} = \left\{ qk : \underset{qk \in Qk'}{\forall} \left[\left[\underset{s \in Wd}{\forall} (qk \not\subset s) \right] \Rightarrow [qk \in Z_{HSd}] \right] \right\}. \qquad (15)$$

Do zbioru Z_{wHSd} należą te elementy zbioru Wh, które należą również do zbioru Z_{HSd}. Zbiór Z_{wHSd} można określić zapisem (16).

$$Z_{wHSd} = \left\{ wh : \underset{wh \in Wh}{\forall} \{ [wh \in Z_{HSd}] \Rightarrow [wh \in Z_{wHSd}] \} \right\} \qquad (16)$$

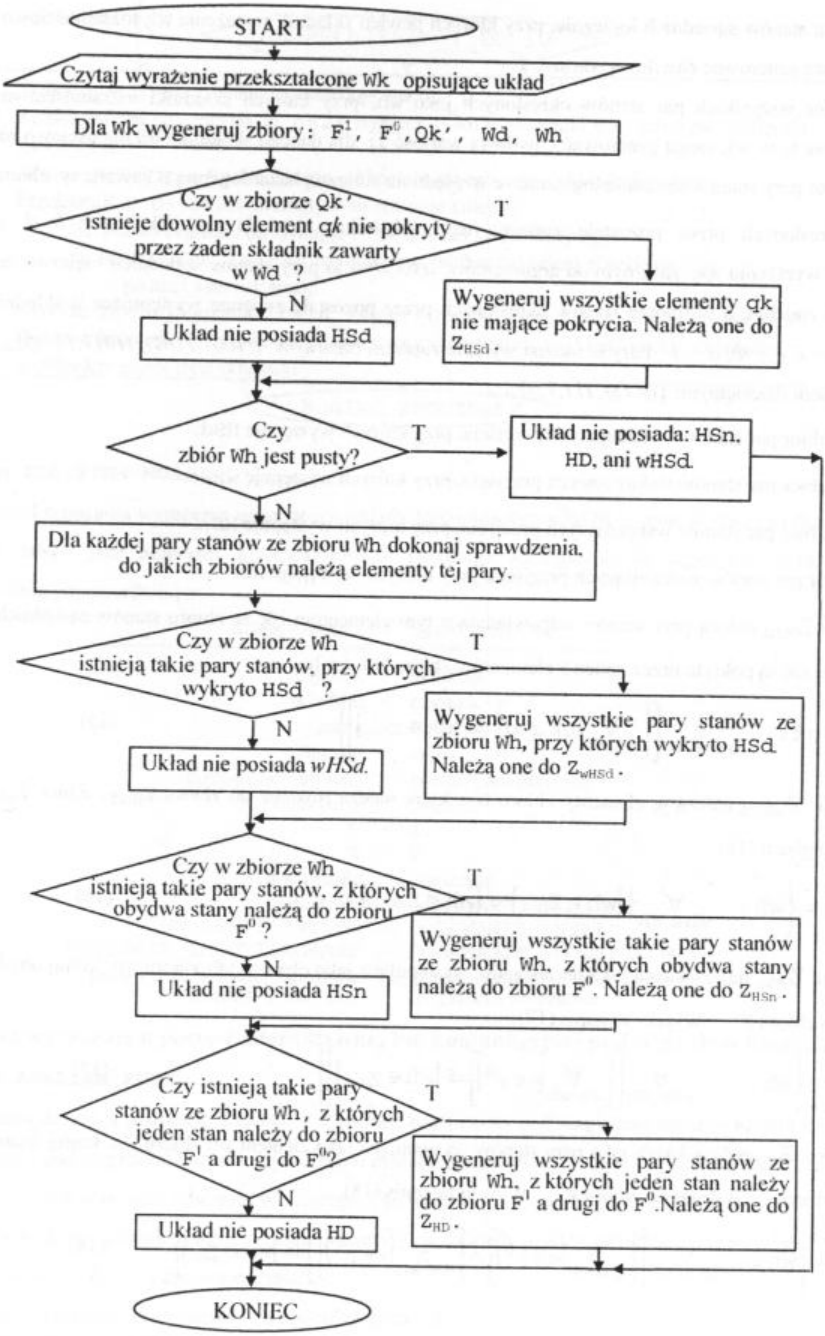
Do zbioru Z_{HSn} należy każda taka para stanów występująca jako element wh zbioru Wh, której obydwa stany e należą do zbioru F^0. Wyraża to zapis (17).

$$Z_{HSn} = \left\{ wh : \underset{wh \in Wh}{\forall} \left[\left[\underset{e \in wh}{\forall} \left[e \in F^0 \right] \right] \Rightarrow [wh \in Z_{HSn}] \right] \right\} \qquad (17)$$

Do zbioru Z_{HD} należy każda taka para stanów występująca jako element wh zbioru Wh, której jeden stan e należy do zbioru F^0, a drugi do zbioru F^1. Wyraża to zapis (18).

$$Z_{HD} = \left\{ wh : \underset{wh \in Wh}{\forall} \left[\left[\underset{e \in wh}{\exists} \left(e \in F^0 \right) \right] \wedge \left[\underset{e \in wh}{\exists} \left(e \in F^1 \right) \right] \right] \Rightarrow [wh \in Z_{HD}] \right\} \qquad (18)$$

[2] Komputerowa analiza wyrażeń uwzględniająca ten algorytm jest dostępna w programie dydaktycznym Anal-wiz na witrynie internetowej: http://zmitac.iinf.polsl.gliwice.pl


START
Czytaj wyrażenie przekształcone Wk opisujące układ
Dla Wk wygeneruj zbiory: F¹, F⁰, Qk', Wd, Wh
Czy w zbiorze Qk' istnieje dowolny element qk nie pokryty przez żaden składnik zawarty w Wd ?
T
N
Wygeneruj wszystkie elementy qk nie mające pokrycia. Należą one do Z_HSd.
Układ nie posiada HSd
Czy zbiór Wh jest pusty?
T
Układ nie posiada: HSn, HD, ani wHSd.
N
Dla każdej pary stanów ze zbioru Wh dokonaj sprawdzenia, do jakich zbiorów należą elementy tej pary.
Czy w zbiorze Wh istnieją takie pary stanów, przy których wykryto HSd ?
T
N
Wygeneruj wszystkie pary stanów ze zbioru Wh, przy których wykryto HSd. Należą one do Z_wHSd.
Układ nie posiada wHSd.
Czy w zbiorze Wh istnieją takie pary stanów, z których obydwa stany należą do zbioru F⁰ ?
T
N
Wygeneruj wszystkie takie pary stanów ze zbioru Wh, z których obydwa stany należą do zbioru F⁰. Należą one do Z_HSn.
Układ nie posiada HSn
Czy istnieją takie pary stanów ze zbioru Wh, z których jeden stan należy do zbioru F¹ a drugi do F⁰?
T
N
Wygeneruj wszystkie pary stanów ze zbioru Wh, z których jeden stan należy do zbioru F¹ a drugi do F⁰. Należą one do Z_HD.
Układ nie posiada HD
KONIEC

Rys. 6-8

Algorytm ANALIZA_ILOCZYNU

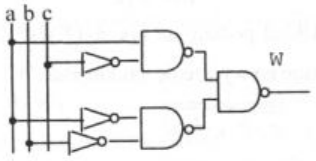
Analogicznie wyrażenie Wk o postaci koniunkcyjnej można ocenić pod względem hazardu algorytmem ANALIZA_ILOCZYNU stosując odpowiedniki:

ANALIZA_SUMY	-	ANALIZA_ILOCZYNU
qk Q_k'	-	qc Q_c'
składnik	-	czynnik
czynnik	-	składnik
1, 0	-	0, 1
HSd, wHSd	-	HSn, wHSn
HSn	-	HSd
F^1, F^0	-	F^0, F^1

6.3. Przykłady analizy układów

Przykład 6.1

Należy przeprowadzić analizę pod względem hazardu dla układu trzypoziomowego NAND o schemacie podanym na rys 6-9.



Rys. 6-9

Układ podany na rys. 6-9 opisuje wyrażenie strukturalne: $W = \overline{\overline{a\,\bar{c}}\ \ \overline{\bar{a}\,b}}$

Po zastosowaniu prawa negacji otrzymujemy: $Wn = a\bar{c} + \bar{a}\,\bar{b}$. Wyrażenie to jest w alternatywnej postaci normalnej (normalnej postaci sumy), ponieważ żaden ze składników wyrażenia nie jest tożsamościowo równy 0. Taka postać jest wolna od HSn i HD. może zawierać jedynie HSd. Na rys. 6-10 przedstawiono obraz wyrażenia dla Z w siatce Karnaugh'a.

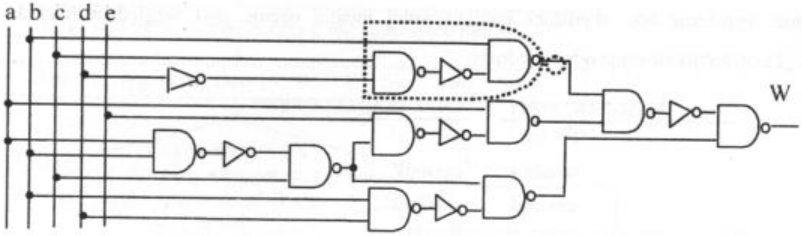
a)	b c	(0) 00	(1) 01	(3) 11	(2) 10
(0)	0	1	1	0	0
(4)	1	1	0	0	1

Wn

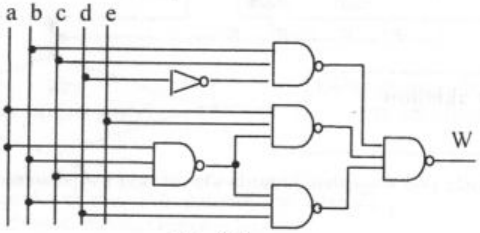
Rys. 6-10

Składnikom badanego wyrażenia w siatce Karnaugh'a odpowiadają grupy jedynkowe. Para sąsiednich logicznie stanów *ek*: $(0,4)_{abc}$ zaznaczona linią przerywaną nie należy do żadnej wspólnej grupy jedynkowej. Zatem przy przejściu między tą parą stanów występuje hazard statyczny w warunkach działania (HSd).

Przykład 6.2



Rys. 6-11



Rys. 6-12

Analizowany jest układ z rys. 6-11. Układ podany na rys. 6-12 jest równoważny logicznie i pod względem hazardu układowi z rysunku 6-11. Opisuje go wyrażenie strukturalne:

$$W = \overline{\overline{b \cdot c \cdot \bar{d}} \cdot \overline{a \cdot e \cdot \overline{a \cdot b \cdot c}} \cdot \ldots \cdot \overline{a \cdot \overline{b \cdot c} \cdot b \cdot d}} \tag{19}$$

Przekształcanie wyrażenia W do postaci alternatywnej Wk

Przekształcenie zgodnie z prawem negacji daje w wyniku postać (20).

$$Wn = \underbrace{b \cdot c \cdot \bar{d}}_{s1} + \underbrace{a \cdot e \cdot (\bar{a} + \bar{b} + \bar{c})}_{s2} + \underbrace{b \cdot d \cdot (\bar{a} + \bar{b} + \bar{c})}_{s3}, \tag{20}$$

Składnik s1 = $b \cdot c \cdot \bar{d}$ nie posiadający czynnika o postaci (c), czyli nie posiadający części cz2 w całości jest przepisywany do Wk.

Pobierany jest kolejny składnik s2. Dla przekształcenia go do postaci alternatywnej wykonywane jest MNOŻENIE. W s2 można wyróżnić części: cz1 i cz2:

$$s2 = \underbrace{a \cdot e \cdot}_{cz1} \underbrace{(\bar{a} + \bar{b} + \bar{c})}_{cz2},$$

W cz1 nie występuje ta sama zmienna w postaci prostej i zanegowanej (zmienna hazardogenna H), ale występuje zmienna *a*, która jako przeciwna występuje w cz2, więc należy zastosować MNOŻENIE_ZMODYFIKOWANE.

Ponieważ cz2 posiada postać alternatywną i nie wymaga redukcji, to: cz2 = cz2s = cz2r.

Pierwszą parę p w procedurze MNOŻENIA_ZMODYFIKOWANEGO stanowią elementy:

- $a \cdot e$ - jako cz1 składnika s2,
- $\bar{a}$ - jako kolejny (pierwszy) składnik wyrażenia cz2r.

Ich iloczyn: I = $a \cdot e \cdot \bar{a}$, jest tożsamościowo równy 0, więc jest realizowane UZUPEŁNIENIE_I_DO_IU.

Wyrażenie U tworzone jest w oparciu o te składniki części cz2r, które nie zawierają zmiennej hazardogennej a, czyli o składniki sU: $\bar{b}$, $\bar{c}$ i posiada postać: U = $\bar{b} \cdot \bar{c} = \overline{b \cdot c}$. Iloczyn I = $a \cdot e \cdot \bar{a}$, zostaje zmodyfikowany do postaci: IU = $a \cdot e \cdot \bar{a} \cdot b \cdot c$. Wynik mnożenia dla składnika s2 tymczasowo przyjmuje postać: Wm = $a \cdot e \cdot \bar{a} \cdot b \cdot c$.

Kolejną parę p stanowią elementy:

- $a \cdot e$ - jako cz1 składnika s2,
- $\bar{b}$ - jako kolejny (drugi) składnik wyrażenia cz2r.

Ich iloczyn: I = $a \cdot e \cdot \bar{b}$, nie jest tożsamościowo równy 0, więc zapisywany jest do Wm.

Podobnie I = $a \cdot e \cdot \bar{c}$, jako iloczyn następnej pary p bez zmian zapisywany jest do Wm.

Ostatecznie wyrażenie Wm uzyskane w wyniku procedury MNOŻENIE dla s2 przyjmuje postać:

$$Wm = a \cdot e \cdot \bar{a} \cdot b \cdot c + a \cdot e \cdot \bar{b} + a \cdot e \cdot \bar{c}. \qquad (21)$$

Wyrażenie Wm dodawane jest logicznie do Wk.

Dla składnika s3 w wyniku podobnego postępowania uzyskuje się wynik mnożenia:

$$Wm = b \cdot d \cdot \bar{a} + b \cdot d \cdot \bar{b} \cdot a \cdot c + b \cdot d \cdot \bar{c} \qquad (22)$$

Końcowy wynik przekształcenia wyrażenia W do postaci alternatywnej przyjmuje postać:

$$Wk = b \cdot c \cdot \bar{d} + a \cdot e \cdot \bar{a} \cdot \underbrace{b \cdot c}_{U} + a \cdot e \cdot \bar{b} + a \cdot e \cdot \bar{c} + b \cdot d \cdot \bar{a} + b \cdot d \cdot \bar{b} \cdot \underbrace{a \cdot c}_{U} + b \cdot d \cdot \bar{c} \qquad (23)$$

W wyrażeniu (23) wskazano wyrażenia U dla iloczynów I tożsamościowo równych 0.

Analiza wyrażenia Wk o postaci alternatywnej

Ocenę wyrażenia Wk pod względem hazardu najlepiej jest zilustrować z wykorzystaniem siatki Karnaugh'a.

Rysunek 6-13 przedstawia obraz wyrażenia Wk w siatce Karnaugha:

c d e	*(0)*	*(1)*	*(3)*	*(2)*	*(6)*	*(7)*	*(5)*	*(4)*
a b	*000*	*001*	*011*	*010*	*110*	*111*	*101*	*100*
(0) *0 0*	0	0	0	0	0	0	0	0
(8) *0 1*	0	0	1	1	1	1	1	1
(24) *1 1*	0	1	1	1	0	0	1	1
(16) *1 0*	0	1	1	0	0	1	1	0

Wk

Rys. 6-13

Zbiory potrzebne w analizie:

F^1 = {*10, 11, 12, 13, 14, 15, 17, 19, 21, 23, 25, 26, 27, 28, 29*}$_{abcde}$

F^0 = {*0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 16, 18, 20, 22, 24, 30, 31*}$_{abcde}$

Qk' = {*(10, 11), (10,14), (10,26), (11,15), (11, 27), (12, 13), (12, 14), (12, 28), (13, 15), (13, 29), (14, 15), (17, 19), (17, 21), (17, 25), 19, 23), (19, 27), (21, 23), (21, 29), (25, 27), (25, 29), (26, 27), (28, 29)*}$_{abcde}$

Wd = {*(10, 11, 26, 27), (10, 11, 14, 15), (12, 13, 28, 29), (17, 19, 21, 23), (17, 19, 25, 27)*}$_{abcde}$

Wh = {*(13, 29), (15, 31), (22, 30), (23, 31)*}$_{abcde}$

W siatce na rysunku 6-13 wymienione zbiory posiadają następujące odpowiedniki:

- elementom zbioru F^1 odpowiadają kratki z wartością 1,
- elementom zbioru F^0 odpowiadają kratki z wartością 0,
- elementom zbioru Wd odpowiadają grupy kratek z wartością 1, zaznaczone obwódką,
- elementom zbioru Wh odpowiadają pary kratek zaznaczone parami strzałek,
- elementom zbioru Qk' odpowiadają pary kratek sąsiednich logicznie z wartością 1.

Do zbioru Z_{HSd} należą cztery pary stanów odpowiadające tym elementom qk ze zbioru Qk', które nie są pokryte przez żaden z elementów zbioru Wd. W siatce na rys. 6-13 odpowiadają im pary stanów 1 sąsiednich logicznie nie należące do żadnej wspólnej grupy jedynkowej.

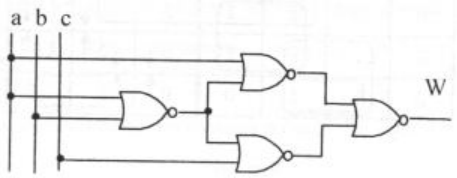
$$Z_{HSd} = \{(12, 14),(13, 15), (21, 29), (25, 29)\}_{abcde}$$

Żadne elementy zbioru Wh nie należą do zbioru Z_{HSd}. Więc zbiór Z_{wHSd} jest pusty. Do zbioru Z_{HSn} należy jedna para stanów ze zboru Wh, której obydwa stany należą do zbioru F^0: $Z_{HSn} = \{(22, 30)\}_{abcde}$. W siatce odpowiada im para sąsiednich logicznie stanów 0 wskazanych parami strzałek.

Do zbioru Z_{HD} należą dwie pary stanów ze zbioru Wh, w których jeden stan należy do zbioru F^1, a drugi do zbioru F^0: $Z_{HD} = \{(15, 31), (23, 31)\}_{abcde}$. W siatce na rys. 6-13 odpowiadają im pary sąsiednich logicznie stanów 0 i 1 wskazanych parami strzałek.

Para stanów $(13, 29)_{abcde}$ ze zbioru Wh, należy do zbioru F^1 i nie należy do Z_{HSd} (należy do wspólnej grupy jedynkowej), więc nie wskazuje ona przejścia hazardogennego.

Przykład 6.3



Rys. 6-14

Analizowany jest układ o schemacie podanym na rys. 6-14. Opisuje gowyrażenie strukturalne:

$$W = \overline{\overline{a+\overline{a+b}} + \overline{c+\overline{a+b}}}. \qquad (24)$$

Przekształcanie wyrażenia W do postaci koniunkcyjnej Wk

Po zastosowaniu prawa negacji otrzymuje się postać (25).

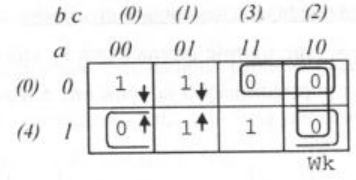
$$Wn = \left(a+\bar{a}\cdot\bar{b}\right)\cdot\left(c+\bar{a}\cdot\bar{b}\right) \qquad (25)$$

Wynik końcowy przekształconego wyrażenia ma postać (26):

$$Wk = \left(a+\bar{a}+b\right)\cdot\left(a+\bar{b}\right)\cdot\left(c+\bar{a}\right)\cdot\left(c+\bar{b}\right) \qquad (26)$$

Analiza wyrażenia (26) pod względem hazardu

Na rys. 6-15 przedstawiono obraz wyrażenia (26) w siatce Karnaugh'a.



Rys. 6-15

Zbiory potrzebne w analizie: $F^0 = \{2, 3, 4, 6\}_{abc}$, $F^1 = \{0, 1, 5, 7\}_{abc}$.

Czynniki wyrażenia (26), które nie są tożsamościowo równe 1: $a + \bar{b}$, $c + \bar{a}$, $c + \bar{b}$, stanowią elementy zbioru Wd. Na rys. 6-15 odpowiadają im grupy zerowe zaznaczone obwódką. W zapisie dziesiętnym zbiór Wd ma postać: $Wd = \{(2,3), (2,6), (4,6)\}_{abc}$.

Zbiór Qc' rozważanej funkcji to zbiór par sąsiednich logicznie elementów zbioru F^0:

$$Qc' = \{(2,3), (2,6), (4,6)\}_{abc}$$

Ponieważ każda z tych par jest pokryta w całości przez przynajmniej jeden element zbioru Wd (grupę zerową), to w badanym układzie nie występuje HSn.

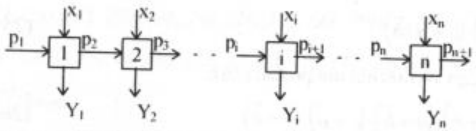
Z czynnika tożsamościowo równego 1: $a + \bar{a} + b$ wynika zbiór par stanów, przy których czynnik ten może generować chwilową wartość 0: $Wh = \{(0,4), (1,5)\}_{abc}$. Na rysunku 6-15 te pary stanów zaznaczono parami strzałek. Wśród par stanów należących do Wh nie ma pary, dla której występuje HSn , zatem w badanym układzie nie ma wHSn.

W zbiorze Wh istnieje taka para stanów: $(1,5)_{abc}$, z której obydwa stany należą do zbioru F^1 rozważanej funkcji. Wskazuje ona, że przy przejściu $(1 \Leftrightarrow 5)_{abc}$ występuje HSd.

W zbiorze Wh istnieje taka para stanów: $(0,4)_{abc}$, z której jeden stan należy do zbioru F^1 a drugi do F^0 rozważanej funkcji. Wskazuje ona, że przy przejściu $(0 \Leftrightarrow 4)_{abc}$ występuje HD.

7. UKŁADY ITERACYJNE

Układy iteracyjne są zbudowane z powtarzających się takich samych typowych dla danego układu podzespołów czyli komórek. Cały układ tworzy się przez odpowiednie połączenie typowych komórek. Rysunek 7-1 przedstawia schemat blokowy układu iteracyjnego o *n* wejściach i *n* wyjściach.



$x_1 \div x_n$ - sygnały wejściowe, $Y_1 \div Y_n$ -sygnały wyjściowe

$p_1 \div p_{n+1}$ - sygnały przeniesienia

Rys. 7-1

Strukturę i-tej komórki znajduje się uwzględniając następujące przesłanki.

W oparciu o wartość sygnału x_i oraz wartość przeniesienia P_i, który informuje ją o tym, jakie wartości posiadają sygnały wejściowe komórek poprzedzających ją, musi ona realizować sygnał wyjściowy Y_i i sygnał przeniesienia P_{i+1}:

$$Y_i = f_i(x_i, p_i) \qquad P_{i+1} = \varphi_{ij}(x_i, p_i).$$

Szczegółowe przesłanki tych zależności zawarte są w słownym opisie warunków pracy układu.

Komórki skrajne można zazwyczaj uprościć uwzględniając:

- brak komórki następnej po komórce n-tej,
- brak komórki poprzedzającej 1-szą komórkę i wynikającą z tego wartość przeniesienia do komórki 1-szej (P_1).

Przykład 7.1

Należy zaprojektować układu o n wej.: $x_1 \div x_n$ i n wyj.: $Y_1 \div Y_n$, dla którego:

$$Y_i = 1 \Leftrightarrow \text{na wejściach: } x_1 \div x_i \text{ jest dwie lub więcej „1".}$$

Sygnał P_i powinien dostarczyć i-tej komórce informacje:

a - na wej: $x_1 \div x_{i-1}$ jest brak „1",

b - na wej: $x_1 \div x_{i-1}$ jest jedna „1",

c - na wej: $x_1 \div x_{i-1}$ jest dwie lub więcej „1".

P_i \ x_i	0	1
a	a	b
b	b	c
c	c	c

P_{i+1}

P_i \ x_i	0	1
a	0	0
b	0	1
c	1	1

Y_i

Rys. 7-2

Dla takich oznaczeń i-tą komórkę opisują siatki: rys. 7-2 Po zakodowaniu trójwartościowego sygnału Pi sygnałami dwójkowymi otrzymujemy siatki podane na rysunku 7-3.

P_i	s_i	t_i
a	0	0
b	0	1
c	1	1

$s_i t_i$ \ x_i	0	1
00	00	01
01	01	11
11	11	11
10	ΦΦ	ΦΦ

$s_{i+1}\, t_{i+1}$

$s_i t_i$ \ x_i	0	1
00	0	0
01	0	1
11	1	1
10	Φ	Φ

Y_i

Rys. 7-3

Oto wyrażenia dla i-tej komórki: $s_{i+1} = s_i + t_i \cdot x_i$, $t_{i+1} = t_i + x_i$, $Y_i = s_{i+1}$.

oraz wartości przeniesienia do 1-szej komórki: $P_1 = a$, czyli: $s_1 = 0$, $t_1 = 0$.

Można uprościć skrajne komórki uwzględniając brak komórki poprzedzającej 1-szą i występującej po n-tej.

dla i = 1: wyrażenia uwzględniające wartości: $s_1 = 0$, $t_1 = 0$ są uproszczone w porównaniu z wyrażeniami ogólnymi dla i-tej komórki: $s_2 = 0$; $t_2 = x_1$; $Y_1 = 0$.

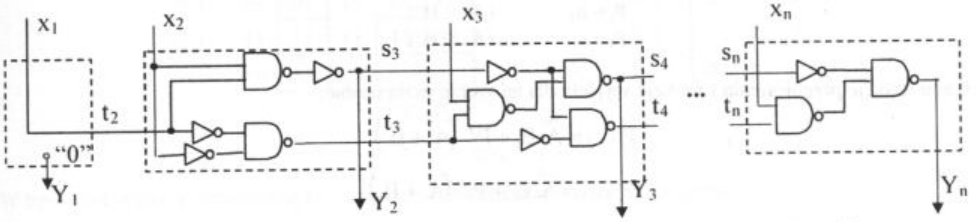
dla i = 2: niektóre wyrażenia uwzględniające wartość: $s_2 = 0$, są również uproszczone w porównaniu z wyrażeniami ogólnymi dla i-tej komórki: $s_3 = t_2 \cdot x_2$, $t_3 = t_2 + x_2$; $Y_2 = s_3$.

Ponieważ sygnały: t_3 oraz s_3 nie mają stałej wartości 0 ani 1, to trzecia komórka (i = 3) jest pełna.

W ostatniej komórce nie trzeba realizować: s_{n+1}, t_{n+1}, więc: dla i = n: $Y_n = s_n + t_n \cdot x_n$.

Ponieważ do realizacji Y_n w n-tej komórce potrzebne są obydwa sygnały przeniesienia, to przedostatnia komórka (i = n-1) jest pełna.

Podsumowując, należy stwierdzić, że komórki o indeksach: 3 ÷ n-1 muszą pozostać bez uproszczenia. Schemat rozwiązania dla przypadku uproszczenia skrajnych komórek przedstawia rys. 7-4.



Komórki: 1, 2 - uproszczone

Komórka 3 pełna

Komórka n-ta uproszczona

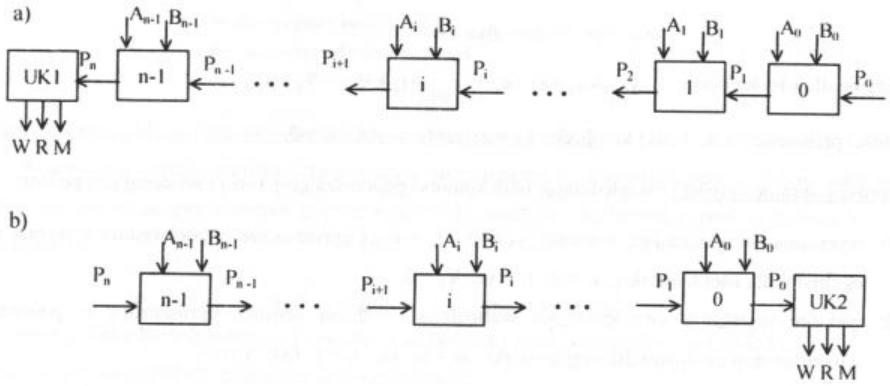
Rys. 7-4

Przykład 7.2

Należy zaprojektować układ umożliwiający porównywanie dwóch n-bitowych liczb dwójkowych (komparator dwójkowy) $A = A_{n-1} \ldots A_1 A_0$ i $B = B_{n-1} \ldots B_1 B_0$, sygnalizując na wyjściach W, R, M odpowiednio:

W R M = 100 jeżeli A > B,
W R M = 010 jeżeli A = B,
W R M = 001 jeżeli A < B.

Przy założeniu struktury iteracyjnej sygnały przeniesienia mogą być generowane w kierunku bitów bardziej znaczących (rys. 7-5a – wersja I) lub mniej znaczących (rys. 7-5b – wersja II).



Rys. 7-5

Na rys. 7-6 pokazano kolejne etapy syntezy i-tej komórki oraz kombinacyjnego układu wyjściowego UK1 komparatora równoległego z sygnałami przeniesienia generowanymi w kierunku bitów bardziej znaczących. W tej wersji sygnał przeniesienia P_i dostarcza informacji o wyniku porównania bitw: 0, 1, ..., (i-2), (i-1)-ego. Rozróżniono wyniki porównania i-bitowych liczb $A' = A_{i-1} A_{i-2} \ldots A_1 A_0$ i $B' = B_{i-1} B_{i-2} \ldots B_1 B_0$:

$P_i = a_1$ $(A' > B')$
$P_i = b_1$ $(A' = B')$
$P_i = c_1$ $(A' < B')$.

Uzyskane funkcje przeniesienia i funkcje wyjścia dla tej wersji mają postać:

$$P'_{i+1} = \overline{A_i}\, B_i + P_i' \left(\overline{A_i} + B_i\right)$$

$$P''_{i+1} = \overline{A_i}\, B_i + P''_i \left(\overline{A_i} + B_i\right)$$

$$W = \overline{P''_n}, \quad R = \overline{P'_n}\, P''_n, \quad M = P'_n$$

Dla komórki 0-wej: $P_0' = 0$, $P_1' = 1$.

a)

P_i \ A_iB_i	00	01	11	10
a_1	a_1	c_1	a_1	a_1
b_1	b_1	c_1	b_1	a_1
c_1	c_1	c_1	c_1	a_1

P_{i+1}

b)

P_i	$P'_i P''_i$
a_1	0 0
b_1	0 1
c_1	1 1

c)

$P'_i P''_i$ \ A_iB_i	00	01	11	10
(a_1) 00	00	11	00	00
(b_1) 01	01	11	01	00
(c_1) 11	11	11	11	00
10				

$P'_{i+1} P''_{i+1}$

d)

$P'_n P''_n$	W	R	M
0 0	1	0	0
0 1	0	1	0
1 1	0	0	1
1 0	Φ	Φ	Φ

Rys. 7-6

Na rys. 7-7 przedstawiono kolejne etapy syntezy i-tej komórki oraz układu wyjściowego UK2 komparatora równoległego z sygnałami przeniesienia generowanymi w kierunku bitów mniej znaczących.

a)

P_{i+1} \ A_iB_i	00	01	11	10
a_2	a_2	a_2	a_2	a_2
b_2	b_2	c_2	b_2	a_2
c_2	c_2	c_2	c_2	c_2

P_i

b)

P_{i+1}	$P'_{i+1} P''_{i+1}$
a_2	0 0
b_2	0 1
c_2	1 1

c)

$P'_{i+1} P''_{i+1}$ \ A_iB_i	00	01	11	10
(a_2) 00	00	00	00	00
(b_2) 01	01	11	01	00
(c_2) 11	11	11	11	11
10				

$P'_i P''_i$

d)

$P'_0 P''_0$	W	R	M
0 0	1	0	0
0 1	0	1	0
1 1	0	0	1
1 0	Φ	Φ	Φ

Rys. 7-7

W tej wersji sygnał przeniesienia P_{i+1} dostarcza informacji o wyniku porównania bitów:

(n-1), (n-2),..... (i+2), (i+1)-ego.

Dla (n-i-2)-bitowych liczb: $A'' = A_{n-1} A_{n-2} \ldots A_{i+2} A_{i+1}$ i $B'' = B_{n-1} B_{n-2} \ldots B_{i+2} B_{i+1}$: rozróżniono następujące wyniki porównania:

$P_{i+1} = a_2$ $(A'' > B'')$,

$P_{i+1} = b_2$ $(A'' = B'')$,

$P_{i+1} = c_2$ $(A'' < B'')$.

Funkcje przeniesienia i funkcje wyjścia dla tej wersji mają postać:

$$P'_i = P'_{i+1} + P''_{i+1}\,\overline{A_i}\,B_i$$

$$P''_i = P'_{i+1} + P''_{i+1}\left(\overline{A_i} + B_i\right)$$

$$W = \overline{P''_o}, \quad R = \overline{P'_o}\,P''_o, \quad M = P'_o$$

Dla komórki (n-1)-szej: $P'_0 = 0$, $P''_0 = 1$.

Należy zwrócić uwagę na fakt, że przy realizacji iteracyjnej czas ustalania wyniku porównania jest stosunkowo duży ze względu na szeregową propagację sygnałów przeniesienia, a stopień złożoności struktury i-tej komórki komparatora jest uzależniony od przyjętego sposobu kodowania.

8. TYPOWE UKŁADY KOMBINACYJNE

Wśród układów kombinacyjnych można wyróżnić takie, które realizują pewne typowe zadania:

- konwertery kodów[1] - układy służące do zamiany informacji przedstawionej w pewnym kodzie binarnym na informację w innym kodzie binarnym:
  - kodery - służą do zamiany informacji w kodzie „1 z n" na kod różny od kodu „1 z n",
  - dekodery - służą do zamiany informacji podanej w kodzie różnym od „1 z n" na kod „1 z n",
  - translatory - służą do zamiany informacji przedstawionej w kodzie różnym od kodu „1 z n" na inny kod różny od kodu „1 z n",
- sumatory[2] - służą do realizacji arytmetycznego dodawania lub odejmowania liczb binarnych,
- komparatory - służą do porównywania liczb binarnych,
- komutatory - służą do przekazywania informacji z wybranego jednego z wielu wejść na wybrane jedno z wielu wyjść,
- generator bitu parzystości - służy do wykrywania błędów powstałych przy przesyłaniu informacji,
- JAL (jednostka arytmetyczno-logiczna) - służy do wykonywania operacji arytmetycznych i logicznych.

Synteza konwerterów kodu została uwzględniona w przykładach:

- 5.2 dla translatora jako układu kombinacyjnego realizowanego na bramkach,
- 12.1 dla translatora zrealizowanego w oparciu o pamięć stałą,
- 14.1 dla translatora zrealizowanego w oparciu o programowane matryce logiczne typu PLA,
- 14.5 dla dekodera zrealizowanego w oparciu o programowane matryce logiczne typu PGA.

Komparator równoległy jest typowym układem iteracyjnym i został uwzględniony w przykładzie 7.2 (komparator szeregowy jako sekwencyjny układ synchroniczny jest uwzględniony w przykładzie 11.4).

Syntezę sumatora i półsumatora jednobitowego przedstawiono jako przykład syntezy układu kombinacyjnego (przykład 5.3) a syntezę tetrady sumatora dziesiętnego pracującego w kodzie 8421 w przykładzie 5.4. (Sumator szeregowy dla liczb dwójkowych jako typowy sekwencyjny układ synchroniczny jest uwzględniony w przykładzie 11.3).

Komutatory są wykorzystywane nie tylko w zakresie zastosowań, dla których zostały „wymyślone" (do przesyłania informacji) ale również do realizacji funkcji logicznych w ogólnym znaczeniu. Dlatego w kolejnym podrozdziale poświęcono im więcej uwagi.

8.1. Komutatory - opis działania i zastosowanie

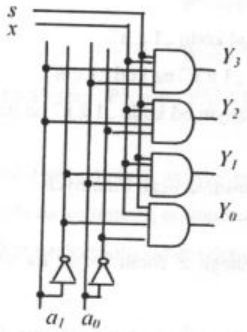
Komutatory (multiplekser, demultiplekser) zwane wybierakami - służą do przekazywania informacji z wybranego jednego z wielu wejść na wybrane jedno z wielu wyjść.

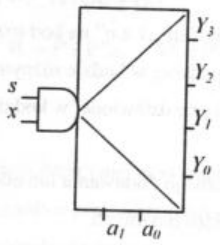
---

[1] Konwertery kodów opisano w ćwiczeniu 2 [6].
[2] Sumatory, a także komparatory i JAL opisano w ćwiczeniu 6 [6].

Demultiplekser

Jest to układ, który służy do przekazywania informacji z jednego wejścia informacyjnego na wybrane adresem jedno z wielu wyjść. Dla demultipleksera 4-bitowego (o czterech wyjściach) o wyjściach prostych rys. 8-1 przedstawia budowę wewnętrzną, symbol oraz wyrażenia realizowane na poszczególnych wyjściach.



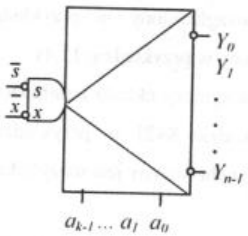


$$Y_0 = \overline{a_1}\,\overline{a_0}\; x\; s$$
$$Y_1 = \overline{a_1}\; a_0\; x\; s$$
$$Y_2 = a_1\,\overline{a_0}\; x\; s$$
$$Y_3 = a_1 a_0\; x\; s$$

x - wejście informacyjne
s - wejście strobujące ze stanem aktywnym „1"
$Y_3 .. Y_0$ - wyjścia
$a_1 .. a_0$ - wejścia adresowe

Rys. 8-1

Demultiplekser n-bitowy (ze względu na łatwiejszą realizacje elementu NAND w porównaniu do AND) w technologii TTL realizowany jest w wersji, która posiada zanegowane wyjścia oraz zanegowane wejścia: strobujące i informacyjne. Symbol i opis znaczenia wyprowadzeń dla demultipleksera n-bitowego o wyjściach zanegowanych podaje rys. 8-2.



$$Y_0 = \overline{a_{k-1} .. a_1 a_0 . x\; s}$$
$$Y_0 = \overline{a_{k-1} .. a_1 a_0 . x\; s}$$
$$\vdots$$
$$Y_{n-1} = \overline{a_{k-1} .. a_1 a_0\; x\; s}$$

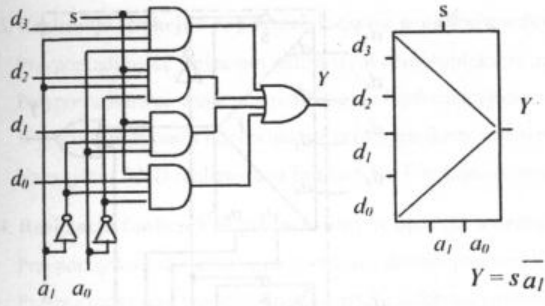
$\overline{x}$ - wejście informacyjne
$\overline{s}$ - wejście strobujące ze stanem aktywnym „0"
$Y_{n-1} .. Y_0$ - wyjścia zanegowane
$a_{k-1} .. a_0$ - wejścia adresowe, gdzie: $n = 2^k$

Rys. 8-2

Każde z wejść adresowych demultipleksera i multipleksera: a_0 , a_1 , a_2 , a_3 ... (czasem oznaczane: A, B, C, D, ...) posiada wagę odpowiednio: $2^0\, 2^1\, 2^2\, 2^3$ ... .

Multiplekser

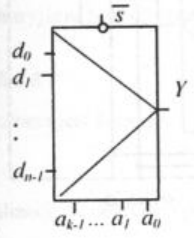
Jest to układ, który służy do przekazywania informacji z wybranego adresem jednego z wielu wejść informacyjnych na jedno wyjście. Dla multipleksera 4-bitowego (o czterech wejściach informacyjnych) rys. 8-3 przedstawia: budowę logiczną, wyrażenie opisujące wyjście oraz symbol. Dla multipleksera n-bitowego o zanegowanym wejściu strobującym rys. 8-4 podaje symbol i opis jego wyprowadzeń oraz wyrażenie realizowane na wyjściu.



s - wejście strobujące ze stanem aktywnym „1"
$d_3 .. d_0$ - wejścia informacyjne
$a_1 a_0$ - wejścia adresowe

$$Y = s\,\overline{a_1}\,\overline{a_0}\,d_0 + s\,\overline{a_1}\,a_0\,d_1 + s\,a_1\,\overline{a_0}\,d_2 + s\,a_1 a_0\,d_2$$

Rys. 8-3

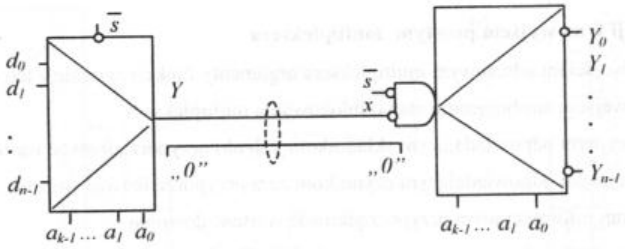


$\overline{s}$ - wejście strobujące ze stanem aktywnym „0"
$d_{n-1} .. d_0$ - wejścia informacyjne
$a_{k-1} .. a_0$ - wejścia adresowe

$$Y = \overline{s}\,\overline{a_{k-1}}\cdots\overline{a_1}\,\overline{a_0}\,d_0 + \overline{s}\,\overline{a_{k-1}}\cdots\overline{a_1}\,a_0\,d_1 + \ldots + \overline{s}\,a_{k-1}\cdots a_1 a_0\,d_{n-1}$$

Rys. 8-4

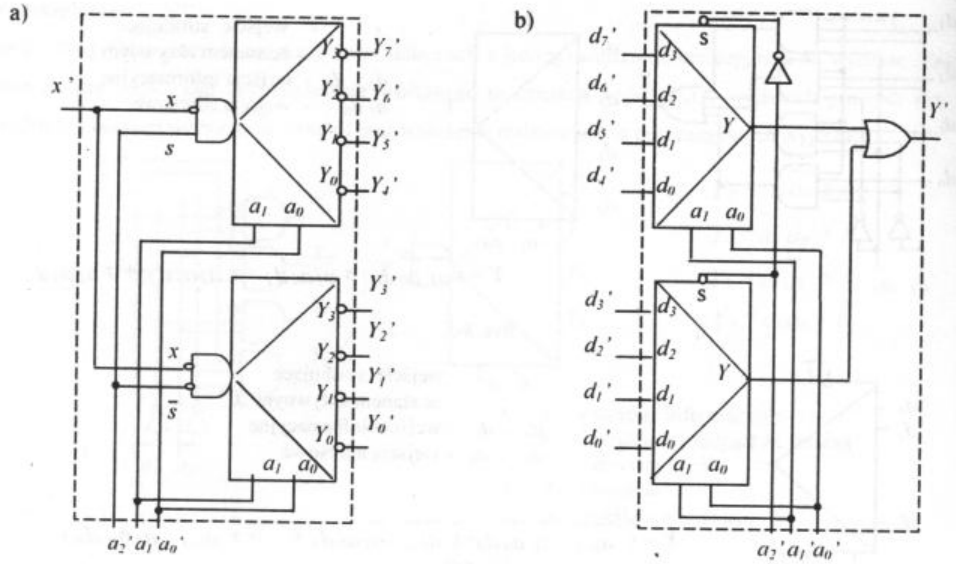
Rysunek 8-5 przedstawia przykład zastosowania komutatorów do przesyłania informacji po jednej parze przewodów z wybranego adresem multipleksera jednego z n wejść w nadajniku od wybranego adresem demultipleksera jednego z n wyjść w odbiorniku.



Rys. 8-5

Wejście strobujące s zarówno w multiplekserze n-bitowym jak i w demultiplekserze n-bitowym może być wykorzystane do budowy układu odpowiadającego $n+1$-bitowego odpowiednio multipleksera lub demultipleksera. Na rysunku 8-6a przedstawiono bardzo prosty sposób realizacji demultipleksera 8-bitowego w oparciu o dwa 4-bitowe demultipleksery. Dzięki temu, że jeden z demultiplekserów składowych posiada wejście strobujące proste a drugi zanegowane, połączenie tych wejść pozwala uzyskać dodatkowe wejście adresowe. Niestety utworzony w ten sposób demultiplekser nie posiada już wejścia strobującego. Dla jego „odbudowy" potrzebny jest dodatkowy układ bramek nie uwzględniony na rys. 8-6a.

Rys. 8-6b przedstawia przykład realizacji 8-bitowego: multipleksera w oparciu o dwa 4-bitowe multipleksery.



Rys. 8-6

Komutatory mają też zastosowanie do realizacji funkcji logicznych[3].

W przypadku, gdy liczba argumentów funkcji nie przekracza liczby wejść adresowych komutatora, synteza jest bardzo prosta i kolejne jej etapy są przedstawione w postaci algorytmów: A1 ÷ A4.

A1. Realizacja funkcji F na wyjściu prostym multipleksera

1. Przyporządkować wejściom adresowym multipleksera argumenty funkcji zgodnie z ich wagami.
2. Przyporządkować wejściu strobującemu stan odblokowania multipleksera.
3. Wejściom informacyjnym odpowiadającym składnikom jedynki przyporządkować wartość „1".
4. Wejściom informacyjnym odpowiadającym czynnikom zera przyporządkować wartość „0".
5. Pozostałym wejściom informacyjnym przyporządkować wartość dowolną.
6. Na wyjściu prostym multipleksera realizowana jest funkcja F w kanonicznej postaci sumy.

A2. Realizacja funkcji F na wyjściu zanegowanym multipleksera

1. Przyporządkować wejściom adresowym multipleksera argumenty funkcji zgodnie z ich wagami.
2. Przyporządkować wejściu strobującemu stan odblokowania multipleksera.
3. Wejściom informacyjnym odpowiadającym składnikom jedynki przyporządkować wartość „0".
4. Wejściom informacyjnym odpowiadającym czynnikom zera przyporządkować wartość „1".
5. Pozostałym wejściom informacyjnym przyporządkować wartość dowolną.
6. Na wyjściu zanegowanym multipleksera realizowana jest funkcja F w kanonicznej postaci iloczynu.

[3] Realizacja funkcji w oparciu o komutatory jest uwzględniona w programie dostępnym w: Mult-syn.zip, na witrynie internetowej: http://zmitac.iinf.polsl.gliwice.pl

A3. Realizacja funkcji F w postaci iloczynu w oparciu o demultiplekser o wyjściach zanegowanych

1. Przyporządkować wejściom adresowym demultipleksera argumenty funkcji zgodnie z ich wagami.
2. Przyporządkować wejściu strobującemu i informacyjnemu stan odblokowania demultipleksera (0).
3. Wyjścia o indeksach odpowiadających czynnikom zera wprowadzić na element AND.
4. Na wyjściu AND realizowana jest funkcja F w kanonicznej postaci iloczynu.

A4. Realizacja funkcji F w postaci sumy w oparciu o demultiplekser o wyjściach zanegowanych

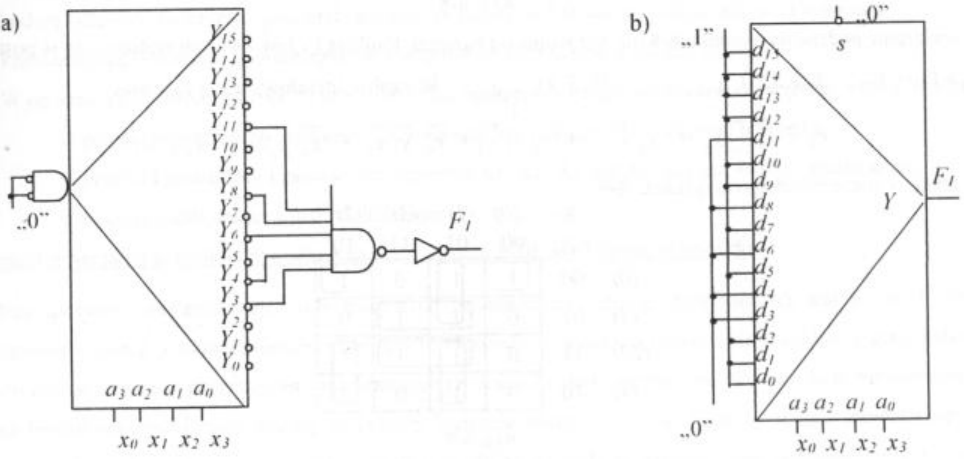
1. Przyporządkować wejściom adresowym demultipleksera argumenty funkcji zgodnie z ich wagami.
2. Przyporządkować wejściu strobującemu i informacyjnemu stan odblokowania demultipleksera (0).
3. Wyjścia o indeksach odpowiadających zanegowanym składnikom jedynki wprowadzić na element NAND.
4. Na wyjściu NAND realizowana jest funkcja F w kanonicznej postaci sumy.

Przykład 8.1

Rozważana jest funkcja:

$$\begin{cases} F_1 = \Sigma(0,1,2,5,7,9,10,13,14,15)_{x_0x_1x_2x_3} \\ F_1 = \Pi(3,4,6,8,11)_{x_0x_1x_2x_3} \end{cases}$$

Jej realizację w oparciu o pojedynczy demultiplekser przedstawia rys. 8-7a. Realizacja ta jest w kanonicznej postaci iloczynu, więc zawiera HSn przy przejściach: $(4\rightarrow6,\ 3\rightarrow11)_{x_0x_1x_2x_3}$. Realizację w oparciu o pojedynczy multiplekser przedstawia rys. 8-7b. Ta realizacja jest w kanonicznej postaci sumy, więc zawiera HSd przy przejściach: $(0\rightarrow1,\ 0\rightarrow2,\ 1\rightarrow5,\ 1\rightarrow9,\ 2\rightarrow10,\ 5\rightarrow7,\ 5\rightarrow13,\ 7\rightarrow15,\ 9\rightarrow13,\ 10\rightarrow14,\ 13\rightarrow15,\ 14\rightarrow15)_{x_0x_1x_2x_3}$.



Rys. 8-7

W oparciu o demultiplekser można też zrealizować F_1 wprowadzając wyjścia demultipleksera o indeksach odpowiadających składnikom jedynki: 0, 1, 2, 5, 7, 9, 10, 13, 14, 15 na końcowy element NAND. Ta realizacja wymagałaby zastosowania 10-wejściowego elementu NAND.

Przykład realizacji funkcji F_1 w oparciu o dwa demultipleksery 8-bitowe przedstawia rys. 8-8a. Funkcja F_1 zależy od czterech zmiennych i nie można jej zrealizować w oparciu o pojedynczy demultiplekser 8-bitowy. Jej postać minimalna może być wyrażona np:

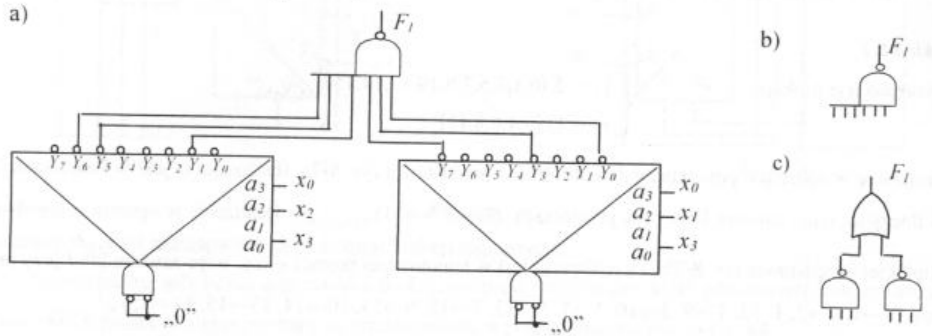
$$F_1 = \overline{x_2}\ x_3 + x_0\, x_2\, \overline{x_3} + \overline{x_0}\ \overline{x_1}\ \overline{x_3} + x_1\, x_3$$

Można ją więc przedstawić jako sumę dwóch funkcji, z których każda zależna jest tylko od trzech zmiennych:

$$F_1 = \left(\overline{x_2}\ x_3 + x_0 x_2\, \overline{x_3}\right) + \left(\overline{x_0}\ \overline{x_1}\ \overline{x_3} + x_1\, x_3\right) = F11 + F12$$

gdzie: $F_{11} = \Sigma(1,5,6)_{x_0x_2x_3}$, $F_{12} = \Sigma(0,3,7)_{x_0x_1x_3}$

Składniki funkcji F_{11} i F_{12} można więc zrealizować w oparciu o osobne demultipleksery. Układ z rys. 8-8c można zastąpić jedną bramką NAND (rys. 8-8b).



Rys. 8-8

Rozwiązanie podane na rysunku 8-8 nie jest wolne od hazardu. Funkcja F_1 jest bowiem realizowana w postaci:

$F_1 = F_{11} + F_{12} = \Sigma(1,5,6)_{x_0x_2x_3} + \Sigma(0,3,7)_{x_0x_1x_3}$ W zapisie literałowym jest to postać:

$$F_1 = \overline{x_0}\,\overline{x_2}\,x_3 + x_0\,\overline{x_2}\,x_3 + x_0\,x_2\,\overline{x_3} + \overline{x_0}\ \overline{x_1}\ \overline{x_3} + \overline{x_0}\ x_1 x_3 + x_0 x_1 x_3.$$

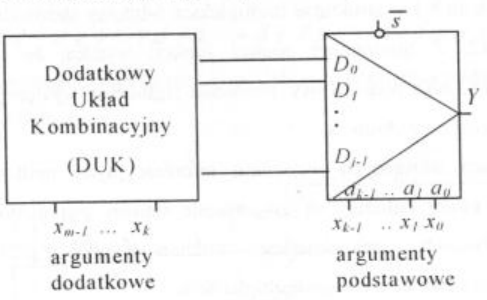
Jej obraz w siatce przedstawia rysunek 8-9.

x_2x_3	(0)	(1)	(3)	(2)
x_0x_1	00	01	11	10
(0) 00	1	1	0	1
(4) 01	0	1	1	0
(12) 11	0	1	1	1
(8) 10	0	1	0	1

Rys. 8-9

Wynika z niego, że są pary sąsiednich logicznie elementarnych implikantów funkcji F_1 $(0\to1,\ 1\to9,\ 2\to10,\ 5\to13,\ 7\to15,\ 14\to15)_{x_0x_1x_2x_3}$, które nie są pokryte przez żaden wspólny implikant rozwiązania według rysunku 8-6. Przy wymienionych przejściach występuje więc HSd.

W przypadku gdy liczba argumentów realizowanej funkcji $F(x_{m-1}, x_m, \ldots, x_k, x_{k-1}, \ldots x_1, x_0)$ przekracza liczbę wejść adresowych multipleksera: $x_{k-1}, \ldots x_1, x_0$ zazwyczaj realizuje się ją w układzie, w którym dodatkowy układ kombinacyjny (rys. 8-10) steruje wejściami informacyjnymi multipleksera. Argumenty: $x_{k-1}, \ldots x_0$ nazywamy wtedy podstawowymi, a argumenty: $x_{m-1}, \ldots, x_k$ dodatkowymi.



Rys. 8-10

Określanie rozwiązania dla Dodatkowego Układu Kombinacyjnego można opisać algorytmem A5[4].

A5. Określenie rozwiązania dla Dodatkowego Układu Kombinacyjnego

1. Wybrać argumenty podstawowe i przyporządkować je wejściom adresowym multipleksera. Aby uzyskać optymalne rozwiązanie DUK, za argumenty podstawowe należy wybrać te, które w minimalnej postaci funkcji występują najczęściej. Wejściu strobującemu $\bar{s}$ przyporządkować wartość 0.
2. Wpisać wartości funkcji do siatki Karnaugh'a zorganizowanej w taki sposób, że argumenty podstawowe kodują kolumny (wiersze) a pozostałe argumenty zwane dodatkowymi kodują wiersze (kolumny).
3. Ponumerować kolumny i wiersze zgodnie z wagami wynikającymi z punktu 1.
4. W oparciu o i-tą kolumnę (wiersz) należy określić funkcję dla i-tego wejścia informacyjnego multipleksera:
   - w postaci minimalnego wyrażenia, gdy ma ona być realizowana w oparciu o bramki,
   - w postaci kanonicznej i zastosować algorytm **A1**, **A2**, **A3** lub **A4**, gdy ma ona być realizowana
   - w oparciu multiplekser lub demultiplekser.

Ocena rozwiązania w strukturze ogólnej według rysunku 8-10 pod względem hazardu

Zaletą sposobu określania wejść informacyjnych multipleksera według algorytmu A5 jest to, że z obrazu wyrażenia w siatce można wnioskować o istnieniu hazardu w realizacji funkcji. Na przykład: z grup jedynkowych zaznaczonych w poszczególnych kolumnach (wierszach) siatki można wnioskować, że w rozwiązaniu istnieje hazard przy przejściach między sąsiednimi logicznie stanami „1" nie należącymi do wspólnej grupy. W rozważanej strukturze nie da się wyeliminować hazardu ze względu na zmianę wejść podstawowych. Łatwo to zauważyć z pomocą siatki, ponieważ grupy tworzone są w osobnych kolumnach (wierszach).

Ze względu na zmianę wejść dodatkowych istnieją te przypadki hazardu, które istnieją w realizacji wejść informacyjnych multipleksera końcowego. Jeżeli wejścia te są realizowane na bramkach, to można określić ich

[4] W ćwiczeniu 11 [6] opisano inny sposoby określania rozwiązania dla takiego przypadku.

realizację w wersji wolnej od hazardu. Jeżeli są one realizowane w oparciu o demultiplekser lub multiplekser, to ich realizacja w postaci kanonicznej sumy (iloczynu) zawiera HSd (HSn) przy przejściach między sąsiednimi logicznie stanami 1 (0), o ile istnieją takie stany sąsiednie logicznie.

Przykład 8.2

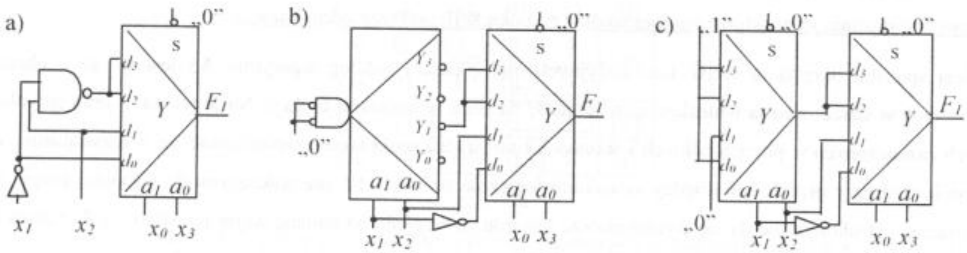
Realizację funkcji F_1 z przykładu 8.1 w strukturze multiplekser 4-bitowy sterowany dodatkowym układem kombinacyjnym podaje rys 8-12. Z minimalnej postaci funkcji wynika, że najczęściej występuje w niej argument x_3 i należy go wybrać jako podstawowy. Pozostałe argumenty występują tak samo często, więc jako drugi argument podstawowy zostaje wybrany x_0.

Dla określenia funkcji sterujących wejściami informacyjnymi multipleksera wpisujemy wartości funkcji do siatki rys. 8-11, której kolumny są zakodowane stanem argumentów podstawowych, a wiersze stanem argumentów dodatkowych. Opis dziesiętny kolumn wynika z przyporządkowania argumentów podstawowych wejściom adresowym końcowego multipleksera.

x_3x_0	(0)	(1)	(3)	(2)
x_1x_2	00	01	11	10
00	1	0	1	1
01	1	1	0	0
11	0	1	1	1
10	0	0	1	1
	d_0	d_1	d_3	d_2

Rys. 8-11

Z wartości funkcji F_1 w poszczególnych kolumnach wynikają wartości funkcji dla wejść informacyjnych multipleksera: $d_0 = \overline{x_1}$, $d_1 = x_2$ $d_2 = d_3 = \overline{\overline{x_1}\, x_2} = \Pi(1)_{x_1x_2}$. Rys. 8-12 podaje wersje realizacji dodatkowego układu kombinacyjnego z zastosowaniem: a - wyłącznie bramek, b - demultipleksera, c- multipleksera.



Rys. 8-12

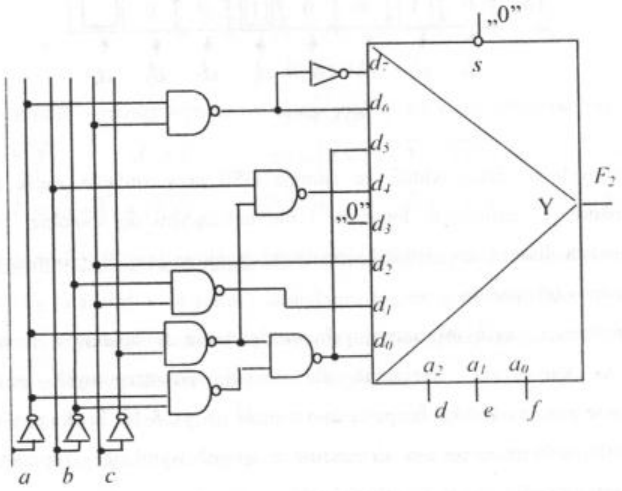
Przykład 8.3
Dana jest funkcja F_2:

$$\begin{cases} F_2 = \Sigma(1, 6, 8, 10, 13, 16, 20, 21, 22, 25, 26, 31, 33, 38, 42, 44, 46, 49, 54, 57)_{abcdef} \\ F_2 = \Pi(2, 3, 7, 14, 23, 24, 27, 28, 29, 30, 32, 34, 35, 37, 39, 40, 41, 47, 50, 51, 52, 53, 56, 59, 60)_{abcdef} \end{cases}$$

Rozważana jest jej realizacja w oparciu o 8-bitowy multiplekser. Z minimalnej postaci tej funkcji:

$$F_2 = \bar{a}\,\bar{b}\,\bar{e} + \bar{a}\,b\,\bar{c}\,\bar{e} + \bar{d}\,\bar{e}\,f\,\bar{c} + \bar{d}\,e\,\bar{f}\,c + d\,e\,\bar{f}\,\bar{c} + d\,e\,f\,\bar{a}\,c + \bar{d}\,\bar{e}\,f\,b + \bar{f}\,d\,\bar{b}\,a$$

wynika, że argumenty: *d, e, f* występują najczęściej, więc należy je przyjąć za podstawowe, a argumenty : *a, b, c* (występujące najrzadziej) jako dodatkowe.



Rys. 8-13

W przypadku realizacji DUK w oparciu wyłącznie o bramki potrzebne są minimalne wyrażenia dla funkcji poszczególnych wejść informacyjnych multipleksera. Rozwiązanie widoczne na rys. 8-13 wynika z siatki podanej na rys 8-14.

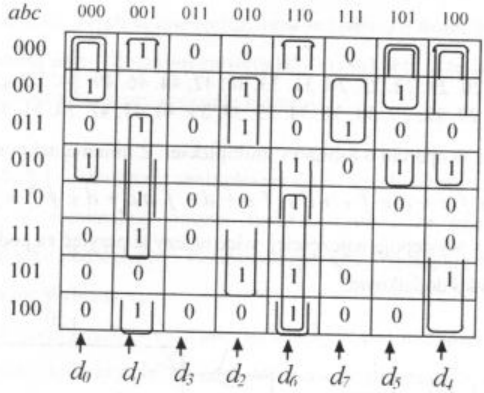
W siatce tej kolumny są zakodowane stanem argumentów podstawowych, a wiersze stanem argumentów dodatkowych. Opis dziesiętny kolumn wynika z przyporządkowania argumentów funkcji wejściom adresowym multipleksera. Opis ten wskazuje indeksy wejść informacyjnych, których funkcje zapisane są w poszczególnych kolumnach.

Celem określenia minimalnych wyrażeń dla funkcji poszczególnych wejść informacyjnych multipleksera, w poszczególnych kolumnach siatki tworzone są grupy. Z grup zaznaczonych na rys. 8-14 w kolumnach wynikają zależności:

$$d_0 = \bar{a}\,\bar{b} + \bar{a}\,\bar{c} \qquad d_2 = c \qquad d_4 = \bar{b} + \bar{c}\,\bar{a} \qquad d_6 = \bar{c} + a = \overline{c\,\bar{a}}$$

$$d_1 = \bar{c} + b = \overline{c\,\bar{b}} \qquad d_3 = 0 \qquad d_5 = d_0 \qquad d_7 = \bar{a}\,c = \overline{d_6}$$

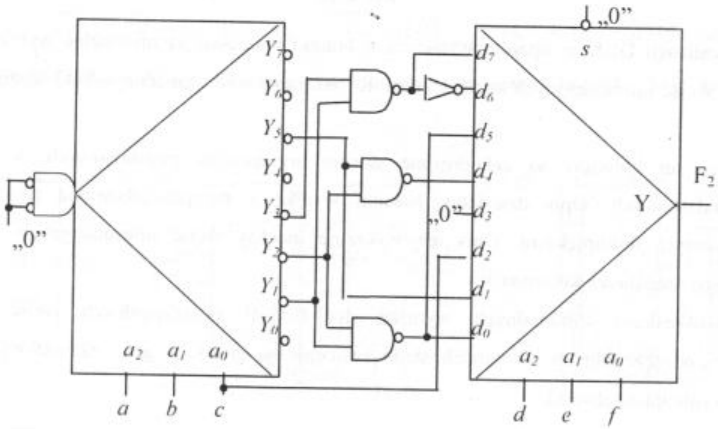
def	0	1	3	2	6	7	5	4



Rys. 8-14

Z grup zaznaczonych w siatce widać, że istnieje HSd przy zmianie wejść podstawowych przy przejściach między stanami „1" sąsiednimi logicznie i nie należącymi do wspólnej grupy 1-kowej. Przy zmianie wejść dodatkowych hazard nie istnieje, ponieważ funkcje wejść informacyjnych multipleksera zrealizowano w sposób wolny od hazardu.

W przypadku realizacji wejść informacyjnych multipleksera w oparciu o demultiplekser i bramki (rys. 8-15) potrzebne są kanoniczne wyrażenia dla funkcji poszczególnych wejść informacyjnych multipleksera. Wyrażenia te można odczytać bezpośrednio z siatki na rys. 8-16. Siatka ta w porównaniu z siatką z rys 8-14 posiada dodatkowo wiersze opisane dziesiętnie w sposób wynikający z przyporządkowania argumentów dodatkowych wejściom adresowym demultipleksera.



Rys. 8-15

d e f	**0**	**1**	**3**	**2**	**6**	**7**	**5**	**4**
abc	000	001	011	010	110	111	101	100

0 000		1	0	0	1	0		
1 001	1	1		1	0		1	
3 011	0	1	0	1	0	1	0	0
2 010	1				1	0	1	1
6 110		1	0	0	1		0	0
7 111	0	1	0					0
5 101	0	0		1	1	0		1
4 100	0	1	0	0	1	0	0	
	↑ d_0	↑ d_1	↑ d_3	↑ d_2	↑ d_6	↑ d_7	↑ d_5	↑ d_4

Rys. 8-16

Z opisu dziesiętnego kolumn i wierszy wynika realizacja funkcji dla wejść informacyjnych multipleksera:

$d_0 = \overline{Y_1} + \overline{Y_2} = \overline{Y_1 Y_2}$ $\quad d_1 = Y_5$ $\quad d_2 = \overline{Y_5} + \overline{Y_3} + \overline{Y_1} = \overline{Y_5 Y_3 Y_2}$ lub $d_2 = c$

$d_3 = 0$ $\quad d_4 = \overline{Y_5} + \overline{Y_2} = \overline{Y_5 Y_2}$ $\quad d_5 = d_0$ $\quad d_6 = Y_1 Y_3$ $\quad d_7 = \overline{Y_3}$ lub $d_7 = \overline{d_6}$

Ocena rozwiązania realizacji F_2 według rys. 8-15. Istnieje HSd przy każdym przejściu między elementarnymi implikantami funkcji F_2 (stanami 1 w siatce) sąsiednimi logicznie ze względu na argumenty podstawowe. Natomiast ze względu na argumenty dodatkowe istnienie hazardu zależy od realizacji wejść d_7, ..., d_0. Ocena zostanie więc dokonana oddzielnie dla poszczególnych wejść.

d_0, d_4, d_5 - realizowane są w kanonicznej postaci sumy, która ogólnie może zawierać HSd, ale rozważane funkcje nie mają sąsiednich logicznie ek więc nie ma problemu HSd (HSn ani HD nie występuje, ponieważ kanoniczna postać sumy jest normalną postacią sumy wolną od tych rodzajów hazardu).

d_1 - realizowana jako jeden czynnik iloczynu jest wolna od hazardu.

d_2 - realizowana jest w sposób wolny od hazardu (jedna grupa jedynkowa na rys. 8-16).

d_3 - jako funkcja stała 0, jest wolna od hazardu.

d_6 - realizowana jest w kanonicznej postaci iloczynu, która zawiera HSn.

d_7 - realizowane jest jako negacja d_6 i posiada HSd jedynie przy przejściu jest między stanem 1 i Φ, który nie ma znaczenia.

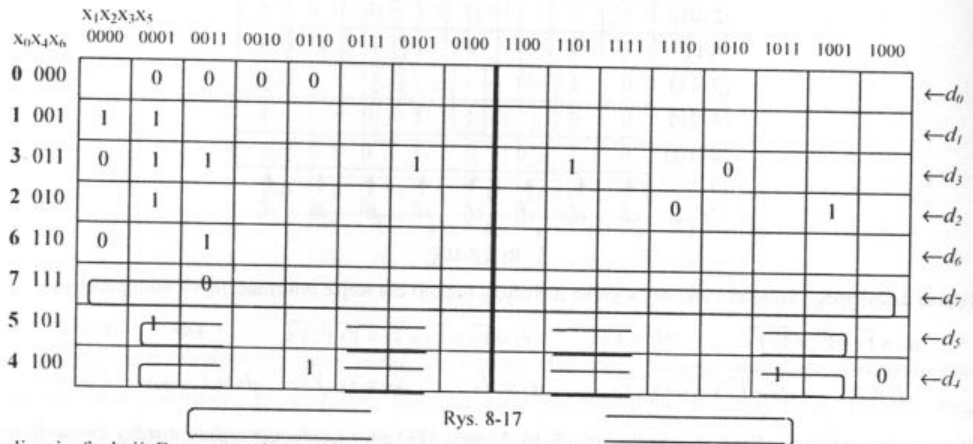
Przykład 8.4

Funkcja F_3 ma następującą postać kanoniczną podaną w zapisie dziesiętnym:

$$\begin{cases} F3 = \Sigma(1, 3, 6, 7, 9, 15, 23, 38, 55, 67, 78, 88, 106)_{x_0 x_1 x_2 x_3 x_4 x_5 x_6} \\ F3 = \Pi(2, 5, 8, 10, 24, 36, 45, 60, 68, 79, 96)_{x_0 x_1 x_2 x_3 x_4 x_5 x_6} \end{cases}$$

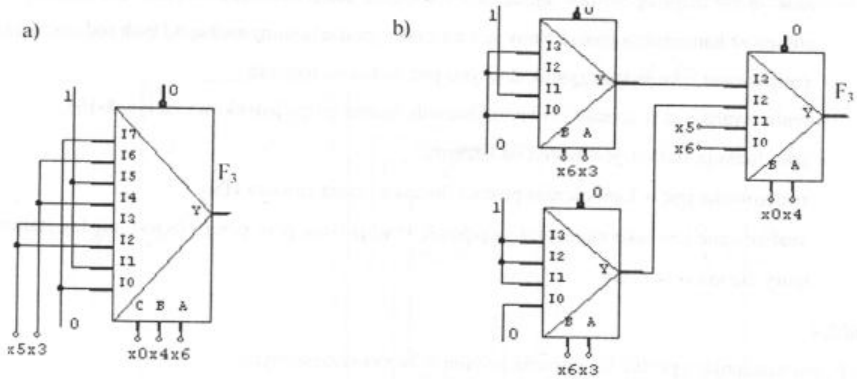
Minimalne wyrażenie alternatywne posiada postać: $F_3 = \overline{x_4}\, x_6 + \overline{x_0}\, x_4 x_5 + x_0 x_3 \overline{x_6}$. Rozważana jest realizacja funkcji w oparciu o multiplekser 8-bitowy. Z minimalnej postaci wyrażenia logicznego wynika, że za podstawowe argumenty należy przyjąć: x_0, x_4, x_6. Rys. 8-18a przedstawia realizację F_3 (obraz ekranu jako wynik komputerowej syntezy za pomocą programu Mult-syn [16]). Siatka na rys. 8-17 uzasadnia rozwiązanie

i pozwala ocenić go pod względem hazardu. Posiada ona wiersze zakodowane kombinacjami argumentów podstawowych, a kolumny kombinacjami argumentów dodatkowych. Widać, że realizacja jest obarczona HSd ze względu na zmianę wejść podstawowych.



Rys. 8-17

Realizację funkcji F_3 w oparciu o wyłącznie 4-bitowe multipleksery (w strukturze drzewo multiplekserów) podaje rysunek 8-18b jako wynik komputerowej syntezy zrealizowanej za pomocą programu Mult-syn [16]. Argumenty: x_0, x_4 przyjęto są jako podstawowe, ponieważ należą one do najczęściej występujących w minimalnej postaci wyrażenia dla F_3. Rozwiązanie dla tej struktury można określić algorytmem A5 lub w sposób opisany w literaturze [6] i [15].

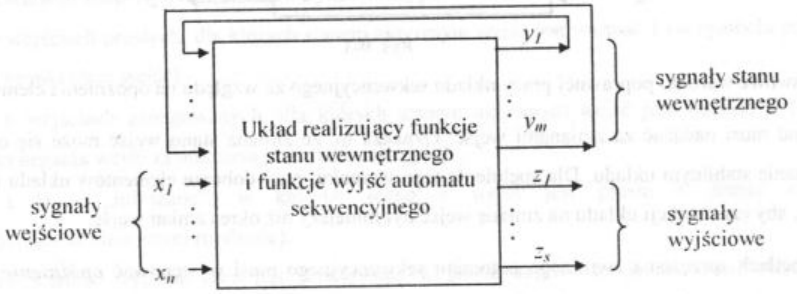


Rys. 8-18

9. ASYNCHRONICZNE STATYCZNE UKŁADY SEKWENCYJNE

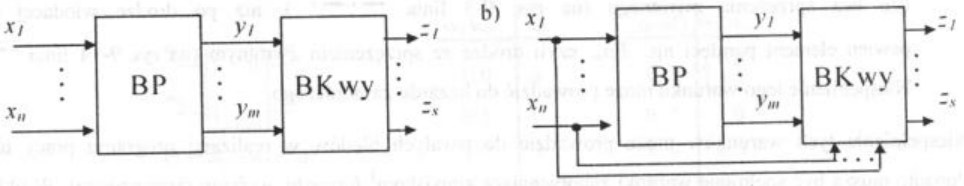
Układ cyfrowy (przełączający) o sygnałach wejściowych $x_1, \ldots, x_n$ i sygnałach wyjściowych $Z_1, \ldots, Z_s$ jest układem sekwencyjnym (z pamięcią) jeżeli posiada przynajmniej jeden taki stan wejść, któremu odpowiada kilka różnych stanów wyjść. To, który z tych stanów pojawi się na wyjściu układu dla zadanego stanu wejść, zależy od poprzednich stanów wejść, czyli od kolejnych zmian sygnałów wejściowych pamiętanych w bloku pamięci (BP) w postaci stanu wewnętrznego układu.

Rysunek 9-1 przedstawia ogólny schemat blokowy asynchronicznego układu sekwencyjnego. Część układu realizująca sygnały stanu wewnętrznego: $y_1, \ldots, y_m$ razem z pętlą sprzężeń zwrotnych nazywana jest Blokiem Pamięci (BP), a część realizująca sygnały wyjściowe: $z_1, \ldots, z_s$ (bez pętli sprzężeń zwrotnych) nazywana jest Blokiem Kombinacyjnym wyjściowym (BKwy). Fragment układu realizujący jeden sygnał stanu wewnętrznego: y_i nazywana jest *i-tym* elementem pamięci (EP_i).



Rys. 9-1

Rys. 9-2 przedstawia schematy układu sekwencyjnego z wyróżnieniem bloków: BP i BKwy. Podstawowe struktury układów sekwencyjnych - to układ Moore'a (rys. 9-2a) i układ Mealye'go (rys. 9-2b). W układzie Moore'a aktualny stan wyjść zależy tylko od aktualnego stanu wewnętrznego. W układzie Mealye'go aktualny stan wyjść zależy od aktualnego stanu wejść i aktualnego stanu wewnętrznego.



$Y(x_1, \ldots, x_n, y_1, \ldots, y_m)$
$Z(y_1, \ldots, y_m)$

gdzie: BP - blok pamięci,
$x_1, \ldots, x_n$ - sygnały wejściowe,
$y_1, \ldots, y_m$ - sygnały stanu wewnętrznego.

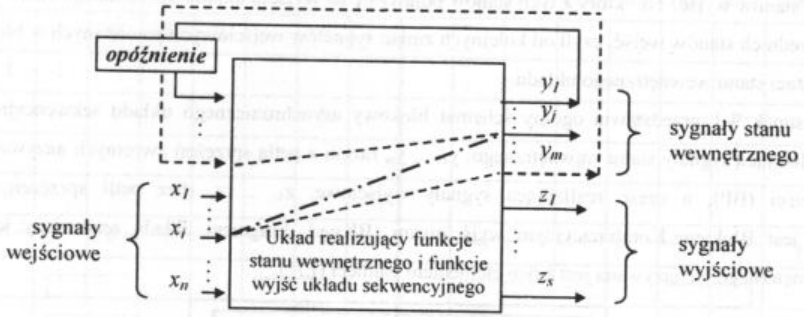
$Y(x_1, \ldots, x_n, y_1, \ldots, y_m)$
$Z(x_1, \ldots, x_n, y_1, \ldots, y_m)$

BKwy - blok kombinacyjny wyjściowy,
$z_1, \ldots, z_s$ - sygnały wyjściowe.

Rys. 9-2

9.1. Podstawowe warunki poprawnej pracy układu sekwencyjnego

Aby asynchroniczny statyczny układ sekwencyjny (rys. 9-3) poprawnie realizował program pracy, muszą być spełnione pewne warunki ze względu na opóźnienia elementów.



Rys. 9-3

Oto podstawowe warunki poprawnej pracy układu sekwencyjnego ze względu na opóźnienia elementów.

1. Układ musi nadążać za zmianami wejść. Oznacza to, że zmiana stanu wejść może się odbywać tylko w stanie stabilnym układu. Dla spełnienia tego warunku, przy doborze elementów układu należy zadbać o to, aby czas reakcji układu na zmianę wejść był mniejszy niż okres zmian wejść.

2. W pętlach sprzężenia zwrotnego automatu sekwencyjnego musi występować ***opóźnienie*** (rys 9-3) aby automat pamiętał stan poprzedni. Ten warunek zazwyczaj jest spełniony i nie ma potrzeby dawania opóźnień w pętlach sprzężenia zwrotnego, ponieważ elementy układu rzeczywistego zawierają opóźnienia. Jednak to opóźnienie może wymagać korekty zgodnie z punktem 3.

3. Każdy z elementów pamięci EP_j realizujący sygnał stanu y_j powinien z mniejszym opóźnieniem reagować na zmianę sygnału wejściowego x_i po drodze wiodącej poza jakimkolwiek elementem pamięci, czyli drodze bez sprzężenia zwrotnego (na rys. 9-3 linia —·—·), niż po drodze wiodącej przez pewien element pamięci np.: Ep_m, czyli drodze ze sprzężeniem zwrotnym (na rys. 9-14 linia - - - -). Niespełnienie tego warunku może prowadzić do hazardu zasadniczego.

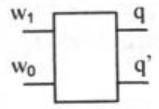
Niespełnienie tych warunków może prowadzić do trwałych błędów w realizacji programu pracy układu. Ponadto muszą być spełnione warunki zapobiegające zjawiskom[1] *hazardu, wyścigu (krytycznego)*. W układach kombinacyjnych spośród trzech wymienionych zjawisk może występować jedynie *hazard*. Zjawisko to opisane zostało w rozdziale 3 oraz 6. W układach sekwencyjnych oprócz hazardu mogą występować zjawiska: *wyścig oraz hazard zasadniczy*, opisane w rozdziale 15.

[1] Dokładniej te zjawiska uwzględnione są w ćwiczeniu nr 7 [6] oraz w programach dydaktycznych dostępnych w: Haz-sym.zip w Haz-stat.zip, H-st-win.zip, Haz-zas.zip na witrynie internetowej: http://zmitac.iinf.polsl.gliwice.pl

9.2. Przerzutniki asynchroniczne statyczne[2]

Najprostszymi układami sekwencyjnymi są przerzutniki asynchroniczne (rys. 9-4)

a)



b)

w_1	w_0	Q
n	n	q
n	a	0
a	n	1
a	a	*

n - stan nieaktywny wejścia
a - stan aktywny wejścia
* - wartość zależna od typu przerzutnika

Rys. 9-4

Pozostałe oznaczenia: w_1 - wejście wpisujące; w_0 - wejście zerujące; q - wyjście główne, czyli stan pracy przerzutnika; Q – funkcja, według której jest realizowany sygnał q, czyli stan wzbudzenia przerzutnika (inaczej: q - jest stanem poprzednim w stosunku do Q); q' - wyjście pomocnicze, które jest negacją wyjścia głównego q pod warunkiem, że przynajmniej jedno z wejść: w_1, w_0 jest w stanie nieaktywnym.

Ze względu na wartość stanu aktywnego wejść wyróżnia się przerzutniki:

- klasy A - o wejściach prostych, dla których stanem aktywnym wejść jest wartość 1 (w symbolu przerzutnika brak znaku kropki obok wejść),
- klasy B - o wejściach zanegowanych, dla których stanem aktywnym wejść jest wartość 0 (w symbolu przerzutnika negacja wejść zaznaczona jest znakiem kropki),
- klasy C i D - „mieszane", w których jedno z wejść jest proste a drugie zanegowane (nie są omówione w niniejszym rozdziale).

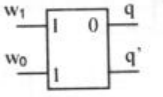
Ze względu na wartość wyjścia, jaka jest generowana w stanie aktywności jednoczesnej obydwu wejść, rozróżnia się przerzutniki: z dominującym wejściem zerującym i z dominującym wejściem wpisującym.

9.2.1. Przerzutniki klasy A

Są nimi przerzutniki o wejściach prostych, czyli takich, dla których stanem aktywnym wejść jest „1"

Przerzutnik o wejściach prostych z dominującym wejściem zerującym

Symbol i wyrażenie



$Q = \overline{w}_0 \ (w_1 + q)$

Tablica przejść

w_1 w_0	Q
0 0	q
0 1	0
1 0	1
1 1	0

Tablica wzbudzeń

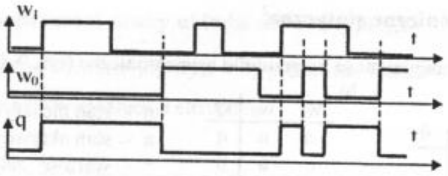
q→Q	w_1 w_0
0 0	0 ϕ
	1 1
0 1	1 0
1 0	ϕ 1
1 1	ϕ 0

Rys. 9-5

Działanie tego przerzutnika ilustruje wykres czasowy podany na rys. 9-6. Na wykresach czasowych ilustrujących działanie przerzutników nie uwzględniono opóźnienia z jakim reaguje wyjście q na zmiany wejść.

---

[2] Omówione dokładniej w ćwiczeniu 3 [6]oraz w programie dydaktycznym dostępnym w Prze-nau.zip na witrynie internetowej: http://zmitac.iinf.polsl.gliwice.pl

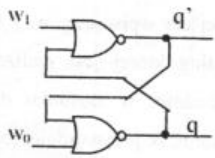


Rys. 9-6

Przerzutnik ten można zrealizować na dwóch bramkach NOR (rys. 9-7).

$$Q = \overline{w_0 + \overline{w_1 + q}}$$

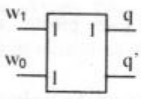
$$Q = \overline{w}_0 \, (w_1 + q)$$



Rys. 9-7

Przerzutnik o wejściach prostych z dominującym wejściem wpisującym

Symbol i wyrażenie



$$Q = w_1 + \overline{w}_0 \, q$$

Tablica przejść

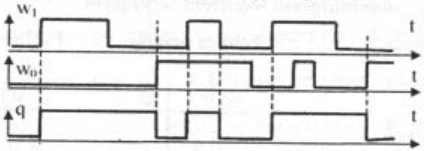
w_1 w_0	Q
0 0	q
0 1	0
1 0	1
1 1	1

Tablica wzbudzeń

q→Q	w_1 w_0
0 0	0 ϕ
0 1	1 ϕ
1 0	0 1
1 1	ϕ 0
	1 1

Rys. 9-8

Działanie tego przerzutnika ilustruje wykres czasowy podany na rys. 9-9.

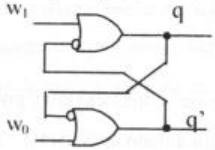


Rys. 9-9

Przerzutnik ten można zrealizować na dwóch elementach implikacji (rys 9-10).

$$Q = \overline{w_1 + w_0 + \overline{q}}$$

$$Q = w_1 + \overline{w}_0 \, q$$



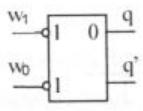
Rys. 9-10

9.2.2. Przerzutniki klasy B

Są to przerzutniki o wejściach zanegowanych, czyli takie, dla których stanem aktywnym wejść jest „0".

Przerzutnik o wejściach zanegowanych z dominującym wejściem zerującym

Symbol i wyrażenie



$Q = w_0 (\overline{w_1} + q)$

Tablica przejść

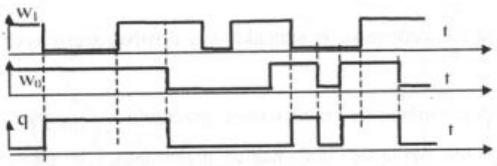
w_1 w_0	Q
1 1	q
1 0	0
0 1	1
0 0	0

Tablica wzbudzeń

q→Q	w_1 w_0
0 0	1 ϕ
	0 0
0 1	0 1
1 0	ϕ 0
1 1	ϕ 1

Rys. 9-11

Działanie tego przerzutnika ilustruje wykres czasowy podany na rys. 9-12.

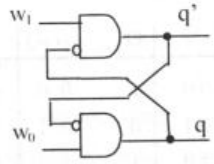


Rys. 9-12

Przerzutnik ten można zrealizować na dwóch elementach zakazu (rys. 9-13).

$$Q = w_0 \cdot \overline{w_1 \cdot \overline{q}}$$

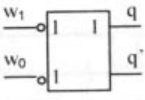
$$Q = w_0 (\overline{w_1} + q)$$



Rys. 9-13

Przerzutnik o wejściach zanegowanych z dominującym wejściem wpisującym

Symbol i wyrażenie



$Q = \overline{w_1} + q\, w_0$

Tablica przejść

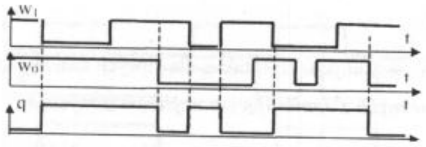
w_1 w_0	Q
1 1	q
1 0	0
0 1	1
0 0	1

Tablica wzbudzeń

q→Q	w_1 w_0
0 0	1 ϕ
0 1	0 ϕ
1 0	1 0
1 1	ϕ 1
	0 0

Rys. 9-14

Działanie tego przerzutnika ilustruje wykres czasowy podany na rys. 9-15.

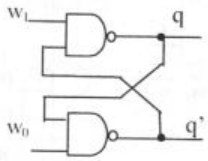


Rys. 9-15

Przerzutnik ten można zrealizować na dwóch bramkach NAND rys. 9-16.

$$Q = \overline{w_1 \cdot \overline{w_0 \cdot q}}$$

$$Q = \bar{w}_1 + q\, w_0$$



Rys. 9-16

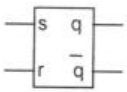
9.2.3. Przerzutniki sr

Gdy przerzutniki klasy A i B są tak sterowane, że stan aktywny obydwu wejść tych przerzutników nie jest możliwy, to $q' = \bar{q}$ oraz:

- obydwa przerzutniki klasy A nie różnią się i mają nazwę: przerzutnik *s r*, gdzie: $w_1 = s$; $w_0 = r$.
- obydwa przerzutniki klasy B nie różnią się i mają nazwę: przerzutnik $\bar{s}\ \bar{r}$, gdzie: $w_1 = \bar{s}$, $w_0 = \bar{r}$.

Przerzutnik s r

Symbol i wyrażenie



$Q = \bar{r}\,(s + q)$

lub $Q = s + q\,\bar{r}$

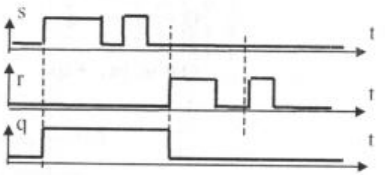
Tablica przejść

s r	Q
0 0	q
0 1	0
1 0	1
1 1	-

Tablica wzbudzeń

q→Q	s r
0 0	0 ϕ
0 1	1 0
1 0	0 1
1 1	ϕ 0

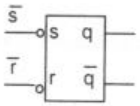
Wykres czasowy



Rys. 9-17

Przerzutnik $\bar{s}\ \bar{r}$.

Symbol i wyrażenie



$Q = r\,(\bar{s} + q)$

lub $Q = \bar{s} + q\, r$

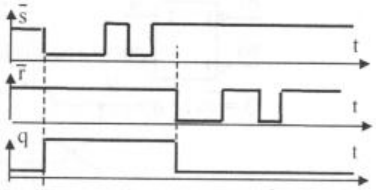
Tablica przejść

$\bar{s}$ $\bar{r}$	Q
0 0	-
0 1	1
1 0	0
1 1	q

Tablica wzbudzeń

q→Q	$\bar{s}$ $\bar{r}$
0 0	ϕ 0
0 1	0 1
1 0	1 0
1 1	0 ϕ

Wykres czasowy

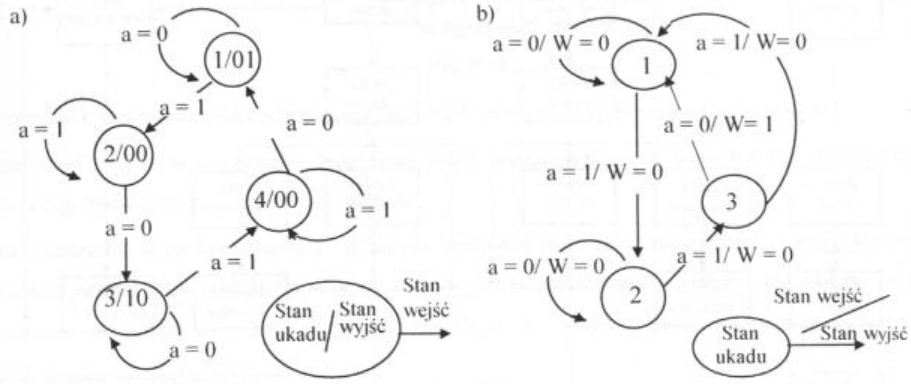


Rys. 9-18

9.3. Synteza asynchronicznych statycznych układów sekwencyjnych metodą siatek przejść (Huffmana)

Punktem wejściowym do tej metody jest algorytm pracy sekwencyjnego układu asynchronicznego przedstawiany najczęściej w postaci: opisu słownego, przebiegów czasowych sygnałów wejściowych i wyjściowych lub grafu przejść.

Graf przejść zawiera wszystkie stany stabilne układu (węzły grafu) oraz przejścia między nimi (gałęzie grafu). Dla układu asynchronicznego pierwotna postać grafu odpowiada zazwyczaj automatowi Moore'a, w którym stany wyjść są jednoznacznie przyporządkowane stanom wewnętrznym układu. Przykład grafu dla układu Moore'a przedstawia rys 9-16a oraz dla układu Mealye'go rys. 9-16b.



Rys. 9-16

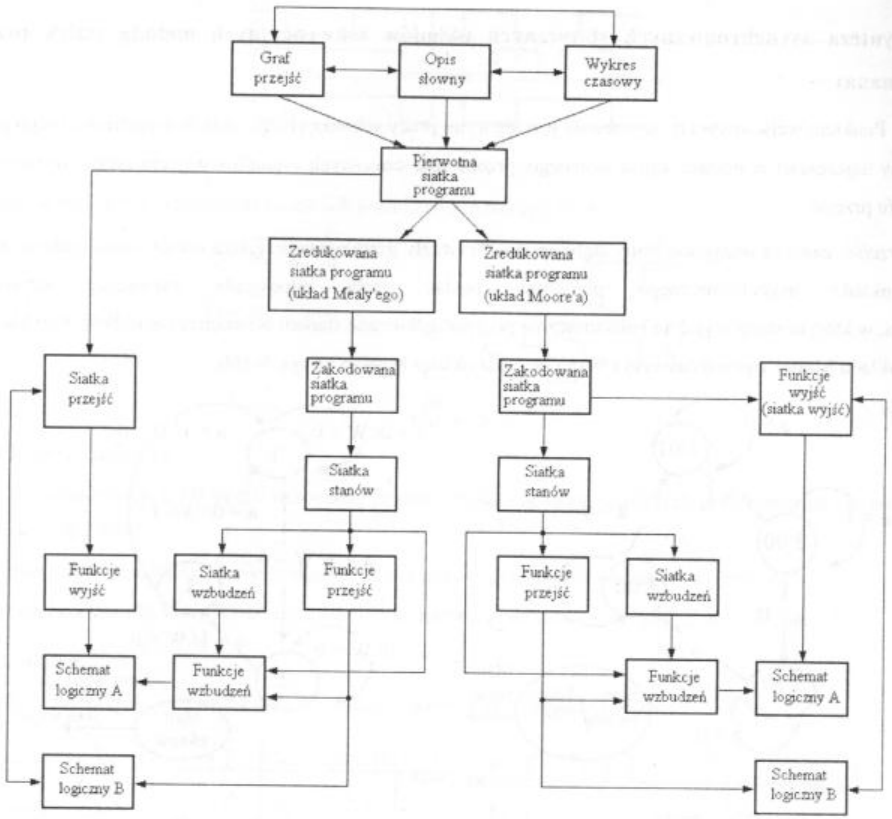
Schemat blokowy kolejnych etapów syntezy w oparciu o metodę siatek przejść przedstawia rys. 9-17. W zależności od rodzaju elementów zastosowanych do realizacji układu w jego wyniku otrzymuje się:

- schemat logiczny A – gdy do realizacji stosuje się oprócz bramek logicznych przerzutniki,
- schemat logiczny B – gdy do realizacji stosuje się wyłącznie bramki logiczne.

W algorytmie można wyróżnić kolejne etapy syntezy[1]:

1. Sporządzenie pierwotnej siatki (tablicy) programu
2. Redukcja pierwotnej siatki (tablicy) programu
3. Kodowanie wierszy zredukowanej siatki (tablicy) programu
4. Określenie funkcji przejść realizowanych przez elementy pamięci
5. Określenie funkcji wyjść

[1] Metoda ilustrowana w programie dydaktycznym dostępnym w Huf-dyd.zip na witrynie internetowej: http://zmitac.iinf.polsl.gliwice.pl



Rys. 9-17

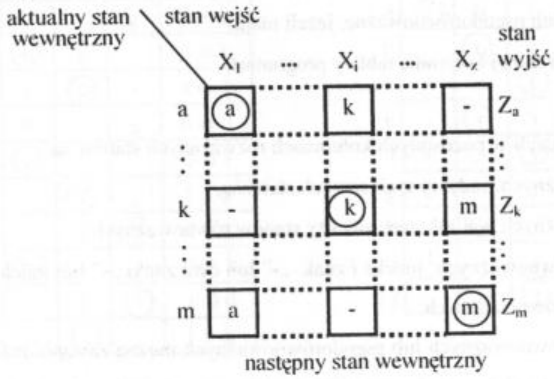
9.3.1. Sporządzanie pierwotnej siatki programu

Pierwotną siatkę programu sporządza się w oparciu o warunki pracy układu podane za pomocą: opisu słownego, wykresu czasowego lub grafu przejść. Ma ona tyle kolumn ile różnych stanów wejść może wystąpić w programie pracy układu oraz tyle wierszy ile stanów wewnętrznych wyróżniamy w opisie pracy układu.

Liczba kolumn siatki rośnie więc wykładniczo ze wzrostem liczby wejść i dla n wejść może mieć maksymalną wartość 2^n. Powoduje to ograniczenie stosowania tej metody do syntezy układów, w których liczba wejść nie jest duża. W takich przypadkach, dla zmniejszenia tego ograniczenia podczas syntezy ręcznej, do etapu, kiedy siatka nie przekształci się w siatkę binarną Karnaugh'a wykorzystywaną do minimalizacji wyrażeń logicznych, można w jej opisie pominąć te kolumny, które odpowiadają stanom wejść nie występującym w algorytmie pracy (stanom Φ układu).

Pierwotna siatka programu odpowiada zawsze automatowi Moore'a. Do kratki określanej przez dany wiersz i kolumnę pierwotnej siatki programu wpisuje się następny stan wewnętrzny, do którego pod wpływem

stanu wejść przypisanego tej kolumnie (zgodnie z programem pracy) przechodzi układ ze stanu stabilnego przypisanego temu wierszowi (rys 9-18).



Rys. 9.18

Poszczególne kratki wiersza odpowiadającego stanowi k pierwotnej siatki programu zawierają:

a) stan stabilny k (k w obwódce) - jeżeli stan wejść przyporządkowany kolumnie rozważanej kratki, nie powoduje zmiany stanu wewnętrznego układu.

b) stan niestabilny m (m bez obwódki) - jeżeli pod wpływem stanu wejść przyporządkowanego kolumnie rozważanej kratki, układ przechodzi ze stanu stabilnego k poprzedniego do stanu m jako następnego po nim; przejście ze stanu niestabilnego m do stabilnego m odbywa się automatycznie, po czasie τ_k wynikającym z opóźnień elementów.

c) znak „-" oznaczający brak przejścia - jeżeli stan wejść przyporządkowany kolumnie rozważanej kratki, jest niemożliwy dla stanu k (program pracy układu nie przewiduje takiego stanu).

Zwykle pierwotną siatkę programu wypełnia się w taki sposób, że najpierw umieszcza się w niej wszystkie stany stabilne wynikające z programu pracy układu, a następnie zaznacza się za pomocą odpowiednich stanów niestabilnych przejścia między stanami stabilnymi, przy czym stan niestabilny m występujący przy przejściu: ⓚ → m → ⓜ umieszcza się w wierszu poprzedniego stanu stabilnego: ⓚ i kolumnie następnego stanu: ⓜ.

9.3.2. Redukcja pierwotnej siatki programu

Redukując liczbę wierszy pierwotnej siatki programu zmniejsza się liczbę elementów pamięci potrzebnych w układzie. Redukcję można przeprowadzić w dwóch etapach:

I - redukcja stanów równoważnych i pseudorównoważnych (redukcja wstępna stanów),

II - redukcja stanów zgodnych.

Etap I – redukcja stanów równoważnych i pseudorównoważnych (redukcja wstępna)

Przebieg I-go etapu redukcji jest niezależny od struktury układu (Moore'a, czy Mealy'ego).

Dwa stany są równoważne lub pseudorównoważne, jeżeli mają:

- zgodne wejścia (są w tej samej kolumnie tablicy programu),
- niesprzeczne wyjścia,
- niesprzeczne przejścia, czyli w pozostałych kolumnach rozważanych stanów są:
  - dla stanów równoważnych: indeksy tych samych stanów,
  - dla stanów równoważnych warunkowo: indeksy stanów równoważnych,
  - dla stanów pseudorównoważnych: indeks i znak „-" lub dwa znaki „-" lub indeksy tych samych stanów, lub indeksy stanów równoważnych.

Grupę n stanów (wierszy) równoważnych lub pseudorównoważnych można zastąpić jednym stanem, gdy każdy stan z każdym innym stanem należącym do rozważanej grupy jest równoważny, równoważny warunkowo lub pseudorównoważny. Podobnie jest w drugim etapie redukcji.

W wyniku redukcji stanów równoważnych lub pseudorównoważnych:

- otrzymuje się stan (wiersz) zastępujący zredukowane stany (wiersze), który posiada:
  - w poszczególnych kratkach (kolumnach) wartości wynikające z nałożenia na siebie redukowanych stanów (wierszy),
  - na pozycji wyjść wartości określone dla każdego z tych wyjść, które posiadają wartość określoną dla przynajmniej jednego ze stanów (wierszy) redukowanych,
- w pozostałych stanach (wierszach) zastępuje się indeksy stanów zredukowanych indeksem stanu, który je reprezentuje po redukcji.

Rys. 9-19a przedstawia przykładowo pary stanów:

- równoważnych: a-d,
- równoważnych warunkowo: b-e (warunek: c≡f) oraz g-i (warunek: f≡j),
- pseudorównoważnych: c-f oraz f-j.

Stanów: b-h oraz e-h nie można zredukować (niezależnie od tego, czy projektujemy układ Moore'a czy Mealy'ego), ponieważ poszczególne stany w tych parach stanów mają sprzeczne wyjścia.

Stany c-j nie są równoważne, ponieważ dla stanu wejść $(01)_{x1x2}$ posiadają one sprzeczne przejścia (przejścia do stanów nierównoważnych: e oraz h). Zatem stan f można zredukować tylko ze stanem c lub ze stanem i (nie można zredukować grupy stanów: c, f, j do jednego stanu).

Siatki na rys. 9-19: b, c, d przedstawiają wyniki redukcji par stanów kolejno: a-d, c-f, b-e. Ponieważ stan f zredukowano ze stanem c i nie zredukowano go ze stanem j, to nie można zredukować stanów g-i (nie jest spełniony warunek: f≡j).

a)

	x_1x_2 00	01	11	10	$Z_1Z_2Z_3$
a	(a)	b	-	g	0 Φ Φ
b	-	(b)	c	-	1 0 1
c	-	e	(c)	-	1 Φ 1
d	(d)	b	-	g	0 Φ 0
e	-	(e)	f	-	Φ 0 1
f	-	-	(f)	g	1 1 Φ
g	d	-	f	(g)	0 0 0
h	a	(h)	f	-	Φ 1 1
i	-	-	j	(i)	0 0 0
j	-	h	(j)	-	1 1 Φ

b)

	x_1x_2 00	01	11	10	$Z_1Z_2Z_3$
a,d	(a)	b	-	g	0 Φ 0
b	-	(b)	c	-	1 0 1
c	-	e	(c)	-	1 Φ 1
e	-	(e)	f	-	Φ 0 1
f	-	-	(f)	g	1 1 Φ
g	a	-	f	(g)	0 0 0
h	a	(h)	f	-	Φ 1 1
i	-	-	j	(i)	0 0 0
j	-	h	(j)	-	1 1 Φ

c)

	x_1x_2 00	01	11	10	$Z_1Z_2Z_3$
a,d	(a)	b	-	g	0 Φ 0
b	-	(b)	c	-	1 0 1
c,f	-	e	(c)	g	1 1 1
e	-	(e)	c	-	Φ 0 1
g	a	-	c	(g)	0 0 0
h	a	(h)	c	-	Φ 1 1
i	-	-	j	(i)	0 0 0
j	-	h	(j)	-	1 1 Φ

d)

	x_1x_2 00	01	11	10	$Z_1Z_2Z_3$
a,d	(a)	b	-	g	0 Φ 0
b,e	-	(b)	c	-	1 0 1
c,f	-	b	(c)	g	1 1 1
g	a	-	c	(g)	0 0 0
h	a	(h)	c	-	Φ 1 1
i	-	-	j	(i)	0 0 0
j	-	h	(j)	-	1 1 Φ

Rys. 9-19

Etap II – redukcja stanów zgodnych (tzw. redukcja wierszy)

Dotyczy ona redukcji stanów stabilnych występujących w różnych kolumnach tablicy programu. Przebieg jej zależy od wyboru struktury automatu: Moore'a lub Mealy'ego.

Dwa stany można zredukować, jeżeli w żadnej z kolumn redukowanych wierszy nie ma indeksów stanów sprzecznych. Stany takie nazywa się zgodnymi. Przy czym :

- gdy rozważane wiersze mają zgodne wyjścia, redukcja dotyczy obu struktur: Moore'a i Mealy'ego,
- gdy wyjścia są sprzeczne, redukcja dotyczy tylko układu o strukturze Mealy'ego.

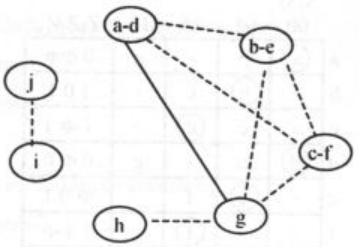
Rys. 9-20 w oparciu o siatkę z rys. 9-19d powtórzoną na rys. 9-20a, ilustruje:

- możliwości redukcji stanów tej siatki dla układu o strukturze Moore'a w postaci linii ciągłych na wykresie redukcyjnym (rys. 9-20b),
- możliwości redukcji stanów dla układu Mealy'ego (linie ciągłe i przerywane na rys. 9-20b),
- wyniki redukcji dla układu o strukturze Moore'a (rys. 9-20c),
- wyniki redukcji dla układu o strukturze Mealy'ego (rys. 9-20d).

a)

	x_1x_2 00	01	11	10	$Z_1Z_2Z_3$
a,d	(a)	b	-	g	0 Φ 0
b,e	-	(b)	c	-	1 0 1
c,f	-	b	(c)	g	1 1 1
g	a	-	c	(g)	0 0 0
h	a	(h)	c	-	Φ 1 1
i	-	-	j	(i)	0 0 0
j	-	h	(j)	-	1 1 Φ

b)



c)

	x_1x_2 00	01	11	10	$Z_1Z_2Z_3$
a,d,g	(a)	b	c	(g)	0 0 0
b,e	-	(b)	c	-	1 0 1
c,f	-	b	(c)	g	1 1 1
h	a	(h)	c	-	Φ 1 1
i	-	-	j	(i)	0 0 0
j	-	h	(j)	-	1 1 Φ

d)

	x_1x_2 00	01	11	10
a,d, b,e, c,f	(a) 0Φ0	(b) 101	(c) 111	g 0Φ0
g, h	a 0Φ0	(h) Φ11	c Φ11	(g) 000
i, j	- ΦΦΦ	h Φ1Φ	(j) 11Φ	(i) 000

Rys. 9.20

W wyniku redukcji stanów zgodnych (wierszy) otrzymuje się wiersz tworzony w sposób zależny od wybranej do realizacji struktury układu.

Dla układu o strukturze Moore'a wiersz ten tworzony jest tak samo, jak w przypadku wstępnej redukcji stanów.

Dla układu o strukturze Mealy'ego wiersz zastępujący zredukowane wiersze zawiera w każdej kratce dwie pozycje: dotyczącą przejścia i dotyczącą stanu wyjść. Pozycja dotycząca przejścia tworzona jest tak samo, jak dla struktury Moore'a.

Pozycja dotycząca stanu wyjść *w kratce stanu stabilnego* zawiera wartości sygnałów wyjściowych odpowiadające temu stanowi stabilnemu. Można je odczytać z wcześniejszych opisów pracy układu np. z: pierwotnej siatki programu, wstępnie zredukowanej siatki programu lub wykresu czasowego.

Pozycja dotycząca stanu wyjść *w kratce stanu niestabilnego* tworzona jest w sposób zależny od wymagań jakie stawiane są dla sygnałów wyjściowych. Dla wyjść można dopuścić przekłamania chwilowe np. wtedy, kiedy steruje on układem o większej bezwładności niż układ, którego synteza jest rozważana. Także nie każdy rodzaj przekłamań na wejściu układu o porównywalnej bezwładności z projektowanym układem, ma znaczenie dla jego pracy.

Jeżeli dla wyjść dozwolone są chwilowe przekłamania, pozycja wyjść w kratce stanu niestabilnego przyjmuje wartość dowolną (Φ). W takim przypadku (konsekwentnie) w realizacji funkcji dla wyjść nie ma potrzeby eliminacji hazardu. W przeciwnym przypadku wartość ta zależy od wartości wyjść w stanach stabilnych sąsiednich w czasie do rozważanego stanu niestabilnego:

- gdy wartość sygnału wyjściowego Z w sąsiednich w czasie stanach jest taka sama: 0 i 0 lub 1 i 1, wtedy w rozważanym stanie niestabilnym należy przyjąć wartość taką samą odpowiednio: 0 lub 1,
- gdy w sąsiednich w czasie stanach wartości sygnału wyjściowego są przeciwne lub dla dowolnego z tych stanów wartość sygnału wyjściowego Z jest nieokreślona (czyli są to wartości: 0 i 1, 0 i Φ, 1 i Φ lub Φ i Φ), wtedy w rozważanym stanie niestabilnym należy przyjąć Φ.

W tym ostatnim przypadku konsekwentnie dala eliminacji chwilowych przekłamań w realizacji funkcji wyjść należy eliminować hazard.

Podobne zasady muszą być spełnione przy określaniu wartości wyjść w zakodowanej siatce programu.

Zasada określania wartości wyjść w stanach niestabilnych zostanie zilustrowana na przykładzie stanu niestabilnego c na rys. 9-20d. Na rysunku tym w stanach niestabilnych przyjęto wartości mające na celu zabezpieczenie sygnałów wyjściowych przed chwilowymi przekłamaniami. Dla wyznaczenia wartości sygnałów wyjściowych w tym stanie należy zbadać ich wartości w stanach sąsiednich w czasie. Stan niestabilny c może wystąpić przy przejściu ⓖ →c→ⓒ albo ⓗ →c→ⓒ .

Z analizy przejścia: ⓖ →c→ⓒ wynika, ze dla Z_1 jest to przejście między poprzednim stanem o wartości 0 i stanem następnym o wartości 1. Z tego przejścia wartość pozycji Z_1 wynika Φ. Z analizy przejścia: ⓗ →c→ⓒ wynika, ze dla Z_1 jest to przejście między poprzednim stanem o wartości nieokreślonej i stanem następnym o wartości logicznej 1. Wartość wynikające z tego przejścia również wynosi Φ. Żadne z przejść przez stan niestabilny c nie narzuca określonej wartości dla Z_1 więc w siatce na tej pozycji jest wartość Φ.

Z podobnej analizy przejścia: ⓖ →c→ⓒ wynika dla Z_2 wartość Φ, ale z analizy przejścia: ⓗ →c→ⓒ między stanami o wartości 1 dla Z_2 wynika wartość 1, dlatego ostateczna wartość na pozycji Z_2 jest 1. Taki sam wynik jest dla Z_3.

9.3.3. Kodowanie wierszy zredukowanej tablicy programu

Wiersze zredukowanej siatki programu (grupy zredukowanych stanów) należy zakodować dwójkowymi (binarnymi) sygnałami stanu wewnętrznego. Kodowanie powinno zapewnić nie tylko rozróżnienie wierszy ale też realizację programu w sposób wolny od wyścigu (patrz rozdział 15.2). Jeżeli liczba wierszy zredukowanej siatki jest zawarta w granicach 2^{n-1} - 2^n, to dla rozróżnienia wierszy wystarczy użyć n sygnałów stanu kodujących wiersze. Dla uniknięcia wyścigu krytycznego czasem trzeba tę liczbę zwiększyć ponad n. Chcąc uniknąć wyścigu, poszczególne wiersze należy zakodować tak, aby sąsiednie w czasie kolejne stany układu były sąsiednimi logicznie (różniły się na jednej tylko pozycji kodu). Poprawne zakodowanie może ułatwić wykres przejść. Na tym wykresie węzły odpowiadają wierszom siatki, a linie łączące wierzchołki grafu wskazują przejścia istniejące w programie między stanami w odpowiadających im wierszach, przy czym:

- linia przerywana odpowiada sytuacji kiedy między stanami w rozważanych wierszach istnieją jedynie przejścia, którymi można pokierować w ramach wierszy siatki, jakie są dla minimalnej liczby sygnałów kodujących je (np. przejście między stanami w wierszach b i c na rys 9-25a),
- linia ciągła odpowiada sytuacji kiedy między stanami w rozważanych wierszach istnieje przynajmniej jedno takie przejście, którym nie można pokierować w ramach wierszy jakie są dla minimalnej liczby sygnałów kodujących je (np. przejście między stanami w wierszach a i c na rys 9-25a).

Zagadnienie bardziej złożonych przypadków kodowania ilustrują przykłady 9.2 oraz 9.4.

9.3.4. Określanie funkcji przejść realizowanych przez elementy pamięci

Na podstawie zredukowanej i zakodowanej siatki programu określa się siatkę stanów wewnętrznych układu i wyznacza wyrażenia dla elementów pamięci. Siatka stanów wewnętrznych układu posiada:

- w stanach stabilnych kod binarny danego wiersza,
- w stanach niestabilnych, kod tego wiersza, do którego układ w najbliższej chwili powinien przejść, z uwzględnieniem kierowania przejściem dla uniknięcia wyścigu krytycznego.

Z siatki stanów wewnętrznych układu wynikają siatki stanów dla poszczególnych elementów pamięci. Z nich wynikają wyrażenia opisujące blok pamięci. Są nimi:

- wyrażenia dla elementów pamięci, czyli funkcje przejść (gdy BP realizowany jest na zasadzie sprzężeń zwrotnych – schematy B na rys. 9.17) albo
- wyrażenia dla funkcji wejść przerzutników, czyli funkcji wzbudzeń przerzutników (gdy BP realizowany jest na przerzutnikach – schematy A na rys. 9.17).

9.3.5. Określanie funkcji wyjść

Dla układu Moore'a

W tej strukturze stan sygnałów wyjściowych zależy tylko od stanu wewnętrznego układu (stanu elementów pamięci). Zależność tę łatwo jest przedstawić w postaci tablicy zależności bezpośrednio w oparciu o zakodowaną tablicę (siatkę) programu. Każdy wiersz tablicy programu jest zakodowany i jest mu przyporządkowany stan wyjść. W przypadku kierowania przejściem dla uniknięcia wyścigu w stanach niestabilnych wynikających z kierowania przejściem należy przyjąć wartość:

1) dowolną (Φ), gdy dopuszczalne są chwilowe przekłamania dla wyjść (wtedy w wyrażeniach dla wyjść nie trzeba eliminować hazardu),
2) wynikającą z analizy przejścia, gdy trzeba wyeliminować chwilowe przekłamania dla wyjść (wtedy w wyrażeniach dla wyjść trzeba eliminować hazard).

Spełnienie warunku w punkcie 2, gdy w kilku kolumnach jest przejście kierowane, może wymagać zwiększenia liczby stanów układu (przez zwiększenie liczby sygnałów stanu). Wtedy należy rozważyć, czy nie korzystniej jest realizować wyjścia w strukturze Mealy'ego.

Dla układu Mealy'ego

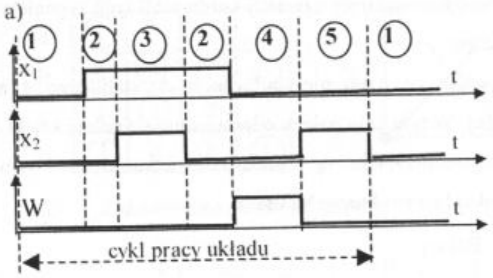
Stan sygnałów wyjściowych zależy od sygnałów wejściowych i sygnałów stanu wewnętrznego układu. Do utworzenia siatki wyjść posługujemy się zredukowaną siatką programu i pierwotną siatką programu. W stanach stabilnych zredukowanej siatki programu należy przyjąć wartości stanów wyjść w tych stanach. Można je odczytać z pierwotnej siatki programu lub wykresu czasowego. Kratki zawierające stany niestabilne wypełnia się podobnie jak dla układu Moore'a:

- gdy dopuszczalne są chwilowe przekłamania to w tych stanach należy przyjąć wartość Φ,
- gdy przekłamania chwilowe należy wyeliminować to:
  - jeśli rozważany stan niestabilny występuje między stanami stabilnymi o takich samych wartościach rozważanego wyjścia, należy dla niego przyjąć wartość wyjścia taką, jaka jest w obydwu stanach stabilnych (w ten sposób dla wyjść otrzymuje się przejście typu 1→1→1 lub 0→0→0),
  - jeśli rozważany stan niestabilny występuje między stanami stabilnymi o różnych wartościach rozważanego wyjścia lub wyjście w którymś ze stanów sąsiednich jest nieokreślone, można dla niego przyjąć wartość Φ (należy zadbać o to aby w przypadku przejścia przez kilka kolejnych stanów niestabilnych zmiana wyjścia następowała jeden raz).

Przykład 9.1

Dla układu opisanego wykresem czasowym (rys. 9-21.a), kolejne początkowe etapy syntezy, wspólne dla struktury Moore'a i Mealy'ego, przedstawiono na rys. 9-21: b, c i d. Etap redukcji wstępnej stanów przeprowadzono na wykresie czasowym (powtarzające się stany pseudorównoważne numer 2).

a)



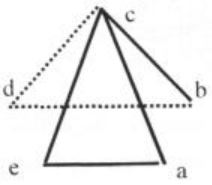
b)

Pierwotna tablica (siatka) programu

	x_1x_2 00	01	11	10	W
a	(1)	-	-	2	0
b	4	-	3	(2)	0
c	-	-	(3)	2	0
d	(4)	5	-	-	1
e	1	(5)	-	-	0

c)

Wykres redukcyjny



redukcja do układu:

—— Moore'a i Mealy'ego

····· Mealy'ego

d)

Możliwość redukcji do minimalnej liczby wierszy			
dla Moore'a		dla Mealy'ego	
wersja 1	wersja 2	wersja 1	wersja 2
a, c, e	a, e	a, c, e	a, e
b	b, c	b, d	b, c, d
d	d		

Rys. 9-21

Dalsze etapy syntezy dla układu o strukturze Moore'a przedstawione są na rys. 9-22 i 9-23, a dla układu o strukturze Mealy'ego na rys. 9-24.

a) ***Zredukowana siatka programu***

	x_1x_2 00	01	11	10
a,c,e	(1)	(5)	(3)	2
b	4		3	(2)
d	(4)	5		

b) ***Kodowanie wierszy***

wiersz	y_1y_2
a,c,e	00
b	01
d	10

c) ***Zakodowana siatka programu***

y_1y_2	x_1x_2 00	01	11	10
00	(1)	(5)	(3)	2
01	4		3	(2)
11				
10	(4)	5		

Binarne siatki stanów wewnętrznych układu:

d) *wspólna*

y_1y_2	x_1x_2 00	01	11	10
00	00	00	00	01
01	11		00	01
11	10			
10	10	00		

Y_1 Y_2

e) *osobne dla poszczególnych elementów pamięci*

y_1y_2	x_1x_2 00	01	11	10
00	0	0	0	0
01	1		0	0
11	1			
10	1	0		

Y_1

y_1y_2	x_1x_2 00	01	11	10
00	0	0	0	1
01	1		0	1
11	0			
10	0	0		

Y_2

f) ***Tablica wyjść***

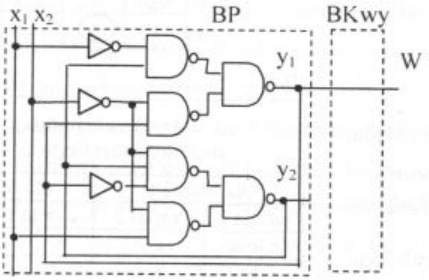
y_1y_2	W
00	0
01	0
11	Φ
10	1

Rys. 9-22

Funkcje przejść: $Y_1 = y_2\bar{x}_1 + y_1\bar{x}_2$ $Y_2 = \bar{y}_1 y_2\bar{x}_2 + x_1\bar{x}_2$ Funkcja wyjść: $W = y_1$

Trzy wiersze zredukowanej siatki programu z rys. 9-22a można zakodować w dowolny sposób dwoma sygnałami stanu wewnętrznego, ponieważ przez wiersz odpowiadający czwartej kombinacji tych sygnałów zawsze można pokierować przejściem dla uniknięcia wyścigu.

Dla uniknięcia wyścigu krytycznego, przejściem ze stanu niestabilnego 4 do stabilnego 4 należy pokierować przez stan $y_1y_2 = 11$ (strzałki na rys. 9-22c). W tym celu należy odpowiednio wypełnić kratki siatki stanów wewnętrznych układu (rys. 9-22d). Pozostałe przejścia są logicznie sąsiednie i nie wymagają kierowania. Rys. 9-23 przedstawia schemat logiczny układu o strukturze Moore'a.



$Y_1 = y_2\bar{x}_1 \quad y_1\bar{x}_2$

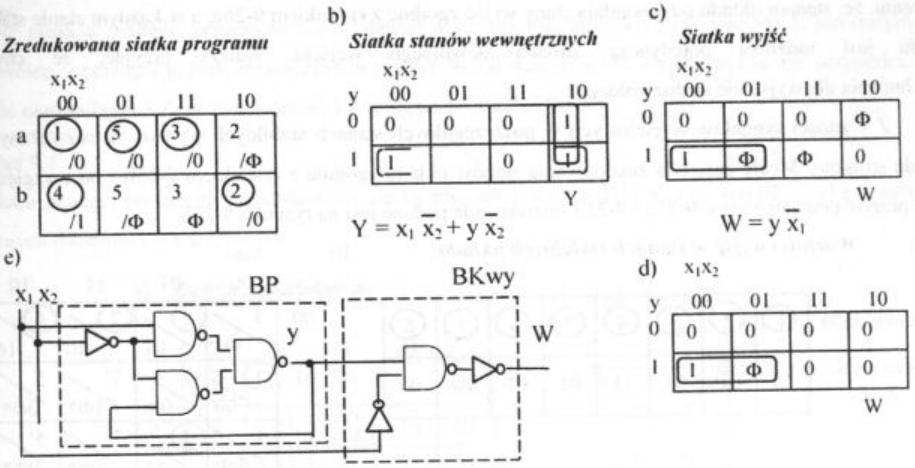
$Y_2 = \bar{y}_1 y_2\bar{x}_2 \quad x_1\bar{x}_2$

$W = y_1$

Rys. 9-23

Dalsze etapy syntezy dla układu o strukturze Mealy'ego przedstawia rys. 9-24. Bez eliminacji chwilowych przekłamań w stanach niestabilnych układu dla wyjścia W można przyjąć wartość Φ jak na rys. 9-24c.

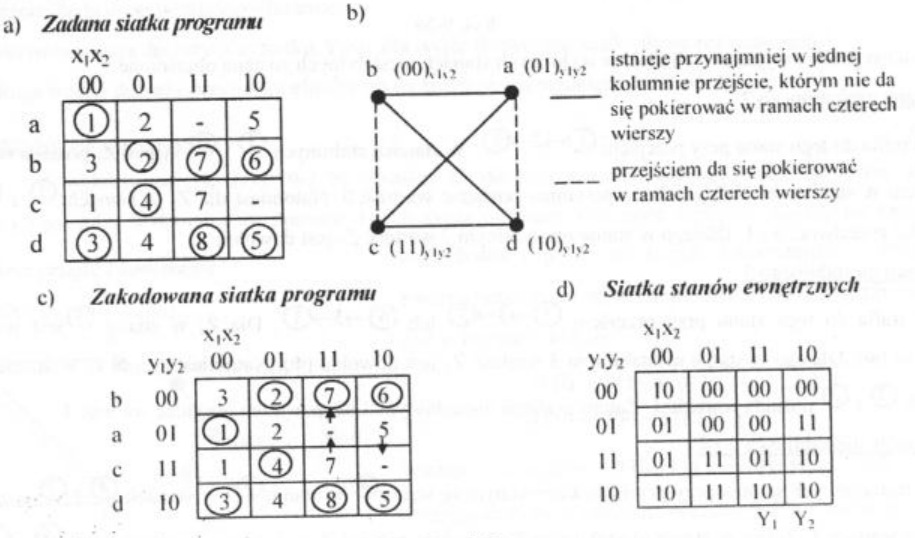
Dla eliminacji chwilowych przekłamań należy przyjąć wartość wynikającą z analizy przejścia, jak na rys. 9-24d. W obydwu przypadkach wyrażenie dla W jest takie samo.



Rys. 9-24

Przykład 9.2

Dla układu opisanego zredukowaną siatką programu (rys. 9-25a) należy określić binarną siatkę stanów.



Rys. 9-25

Dla rozróżnienia czterech wierszy trzeba i wystarczy dwa sygnały stanu. Wykres przejść i kodowania (rys. 9-25b) pomaga ocenić, czy nie zwiększając tej liczby można określić rozwiązanie wolne od wyścigu krytycznego. W rozważanym przypadku wiersze można zakodować dwoma sygnałami stanu: y_1 y_2 i pokierować przejściami niesąsiednimi logicznie: 7 →⑦ oraz 5 →⑤ tak, jak podają rysunki: 9-25c i 9-25d.

Przykład 9.3

Dla układu opisanego zredukowaną siatką programu podaną na rys. 9-25a, należy określić siatkę dla wyjść, przy założeniu, że stanom układu odpowiadają stany wyjść zgodnie z rysunkiem 9-26a, a w każdym stanie stabilnym układu jest możliwa pojedyncza zmiana dowolnego wejścia. Należy przyjąć, że chwilowe przekłamania dla wyjść nie są dozwolone.

Z wartości sygnałów wyjściowych w poszczególnych stanach stabilnych wynika, że rozważany układ posiada strukturę Mealy'ego. Dla zakodowania stanów układu zgodnie z rysunkiem 9-25b i po uwzględnieniu siatki przejść podanej na rys. 9-25c i 9-25d, rozwiązanie podane jest na rysunku 9-26b.

a) ***Wartości wyjść w stanach stabilnych układu***

stan układu	①	②	③	④	⑤	⑥	⑦	⑧
stan wyjść $Z_1 Z_2$	00	01	Φ1	11	01	1Φ	10	00

b)

y_1y_2 \ x_1x_2	00	01	11	10
00	3 / Φ1	② / 01	⑦ / 10	⑥ / 1Φ
01	① / 00	2 / 0Φ	7' / 1Φ*	5 / 0Φ**
11	1 / ΦΦ	④ / 11	7 / 1Φ*	5' / 0Φ**
10	③ / Φ1	4 / Φ1	⑧ / 00	⑤ / 01

stan układu / Z_1Z_2

Rys. 9-26

Wartości sygnałów wyjściowych Z_1Z_2 w wybranych stanach niestabilnych zostaną objaśnione.

Dla stanu niestabilnego 2

Układ trafia do tego stanu przy przejściu ①→2→②. W stanach stabilnych ① i ② sygnał Z_1 posiada wartość 0. Zatem w stanie niestabilnym 2 też powinien posiadać wartość 0. Natomiast dla Z_2 w stanach ① i ② są wartości przeciwne: 0 i 1. Dlatego w stanie niestabilnym 2 wartość Z_2 jest dowolna.

Dla stanu niestabilnego 3

Układ trafia do tego stanu przy przejściu ②→3→③ lub ⑥→3→③. Dla Z_1 w stanie ③ jest wartości dowolna (Φ). Dlatego w stanie niestabilnym 3 wartość Z_1 jest dowolna (Φ). Natomiast sygnał Z_2 w stanach stabilnych ② i ③ posiada wartość 1. Zatem w stanie niestabilnym 3 też powinien posiadać wartość 1.

Dla stanów niestabilnych 7 i 7'

Układ trafia do tych stanów przy przejściu kierowanym ze stanu ④ do stanu ⑦. W stanach ④ i ⑦ sygnał Z_1 posiada wartość 1. Zatem w stanie niestabilnym 7 i 7' należy przyjąć $Z_1 = 1$. Natomiast w stanach ④ i ⑦ Z_2 ma wartości przeciwne:: 1 i 0. Więc w stanach niestabilnych 7 i 7' wartość Z_2 jest dużym stopniu dowolna, ale z pewnym ograniczeniem (na rys. 9-26b wartość Φ*). Należy mianowicie zadbać o to, aby przy rozważanym przejściu sygnał Z_2 zmienił wartość tylko jeden raz, należy więc wykluczyć przypadek kiedy w stanie niestabilnym 7 Z_2 posada wartość 0 a w stanie 7' wartość 1.

Dla stanów niestabilnych 5 i 5'

Układ trafia do tych stanów przy przejściu kierowanym ze stanu ① do stanu ⑤. W stanach: ① i ⑤ Z_1 posiada wartość 0. Zatem w stanie niestabilnym 5 i 5' należy przyjąć $Z_1 = 0$. Natomiast Z_2 w stanach: ① i ⑤ posiada wartości przeciwne: 0 i 1. Dlatego w stanach niestabilnych 5 i 5' wartość Z_2 jest dużym stopniu dowolna, z pewnym jednak ograniczeniem (na rys. 9-26b wartość Φ**). Należy wykluczyć przypadek kiedy w stanie niestabilnym 5 Z_2 posada wartość 1 a w stanie 5' wartość 0.

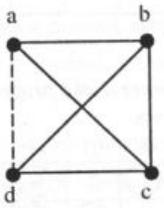
Przykład 9.4

Dla bloku pamięci opisanego zredukowaną siatką programu rys. (9-27a) należy określić siatkę stanów wewnętrznych (binarną) i wartości wyjść.

a) ***Zadana siatka programu***

	x_1x_2 00	01	11	10	Z_1Z_2
a	①	2	⑨	5	00
b	3	②	⑦	⑥	01
c	1	④	7	5	11
d	③	4	⑧	⑤	01

b) ***Wykres przejść***



Rys. 9-27

Rozważone będą dwie wersje rozwiązania:

- pierwsza wersja dotyczy przypadku, kiedy dla wyjść dozwolone są chwilowe przekłamania,
- druga wersja dotyczy przypadku eliminowania chwilowych przekłamań dla wyjść.

Pierwsza wersja rozwiązania

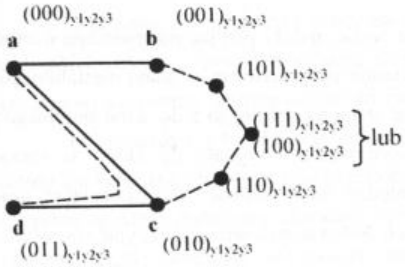
Z wykresu przejść (rys. 9-27b) wynika, że do zakodowania wierszy trzeba użyć trzy sygnały stanu: $y_1y_2y_3$, gdyż kodowanie za pomocą dwóch sygnałów stanu nie może zapewnić sąsiedztwa logicznego wierzchołka (np. b) do trzech z pozostałych. Przejściem z wierzchołka *c* do wierzchołka *b* (rys. 9-28) należy pokierować przez stany kolejno:

$(110 \rightarrow 111 \rightarrow 101)\ y_1y_2y_3$ lub

$(110 \rightarrow 100 \rightarrow 101)\ y_1y_2y_3$.

Wybrano przejście: $(110 \rightarrow 111 \rightarrow 101)\ y_1y_2y_3$ tak, jak to ilustrują strzałki w siatce na rysunku 9-29a i siatka binarna na rysunku 9-29b. Przejściem z wiersza a do d należy pokierować przez wiersz c (linia zagięta przerywana na rys. 28).

Wykres przejść i kodowania



Rys. 9-28

Wartości wyjść w wierszach dotyczących przejść kierowanych są dowolne w tej wersji rozwiązania, ponieważ dotyczą stanów niestabilnych.

a)

Zakodowana siatka programu

	$y_1y_2y_3$	x_1x_2 00	01	11	10	Z_1Z_2
(a)	000	(1)	2	(9)	5	00
(b)	001	3	(2)	(7)	(6)	01
(d)	011	(3)	4	(8)	(5)	01
(c)	010	1	(4)	7	5	11
	110					
	111					
	101					
	100					

b)

Siatka stanów wewnętrznych

$y_1y_2y_3$	x_1x_2 00	01	11	10	Z_1Z_2
000	000	001	000	010	00
001	011	001	001	001	01
011	011	010	011	011	01
010	000	010	110	011	11
110			111		ΦΦ
111			101		ΦΦ
101			001		ΦΦ
100					ΦΦ

$Y_1Y_2Y_3$

c)

Zakodowana siatka programu

	$y_1y_2y_3$	x_1x_2 00	01	11	10	Z_1Z_2
(a)	000	(1)	2	(9)	5	00
(b)	001	3	(2)	(7)	(6)	01
(d)	011	(3)	4	(8)	(5)	01
(c)	010	1	(4)	7	5	11
	110					Φ1
	111					01
	101					01
	100					0Φ

d)

Siatka stanów wewnętrznych

$y_1y_2y_3$	x_1x_2 00	01	11	10	Z_1Z_2
000	000	001	000	100	00
001	011	001	001	001	01
011	011	010	011	011	01
010	000	010	110	011	11
110			111		Φ1
111			101	011	01
101			001	111	01
100				101	0Φ

$Y_1Y_2Y_3$

Rys. 9-29

Druga wersja rozwiązania

Chcąc uzyskać rozwiązanie wolne od chwilowych przekłamań dla wyjść, należy przyjąć odpowiednie wartości w stanach dolnej połowy siatki z rys.: 9-29c i 9-2d. przez które kieruje się przejściem ze stanu niestabilnego 7 do stanu stabilnego 7. Ponadto należy zmienić drogę kierowania ze stanu niestabilnego 5 do stanu stabilnego 5. Nie zmieniając tej drogi nie będzie można zapewnić poprawnych wartości sygnału Z_1, który w stanach wymienionych przyjmuje wartość 0, a w stanie pośrednim między nimi wartość 1. Zmieniając drogę kierowania przejściem, można w stanach niestabilnych (dotyczących dodatkowych wierszy) przyjąć wartości zapewniające pracę bez chwilowych przekłamań. Poprawne rozwiązanie dla rozważanej wersji podają rysunki: 9-29c i 9-29d.

9.4. Opis metody tablic kolejności łączeń (Siwińskiego)

Według tej metody [14] w oparciu o: opis słowny programu działania układu, wykres czasowy lub graf przejść sporządza się opis w postaci tablicy kolejności łączeń. Powinien on uwzględniać wszystkie rodzaje pracy układu.

Tablica kolejności łączeń (TKŁ) zawiera wiersze dotyczące sygnałów wejściowych i sygnałów wyjściowych zwanych elementami wejściowymi i elementami wyjściowymi oraz kolumny przeznaczone na zaznaczenie kolejnych zmian tych sygnałów. Każda zmiana rozdziela dwa takty. Oprócz wymienionych, tablica posiada dodatkowe wiersze: *Takty* - przeznaczony na zaznaczenie numerów kolejnych taktów oraz *Stan układu* - przeznaczony na dziesiętny zapis numerycznego stanu układu uwzględniający wagi przypisane poszczególnym sygnałom.

W takcie pierwszym zapisany jest stan wszystkich sygnałów za pomocą znaków: „+" oznaczającego wartość logiczną 1 oraz „-" oznaczającego wartość logiczną 0. W następnych taktach zaznaczany jest tylko stan sygnałów zmienionych.

Dla poszczególnych elementów wyjściowych można określić warunki działania i warunki niedziałania. *Warunki działania elementu* zaczynają istnieć w takcie poprzedzającym zmianę tego elementu na „+" i przestają istnieć w takcie poprzedzającym zmianę tego elementu na „-" (linia ciągła na rysunku np. 9-31). Pozostałe takty w cyklu pracy układu należą do *warunków niedziałania* tego elementu (zaznaczone linią przerywaną na rysunku np. 9-31).

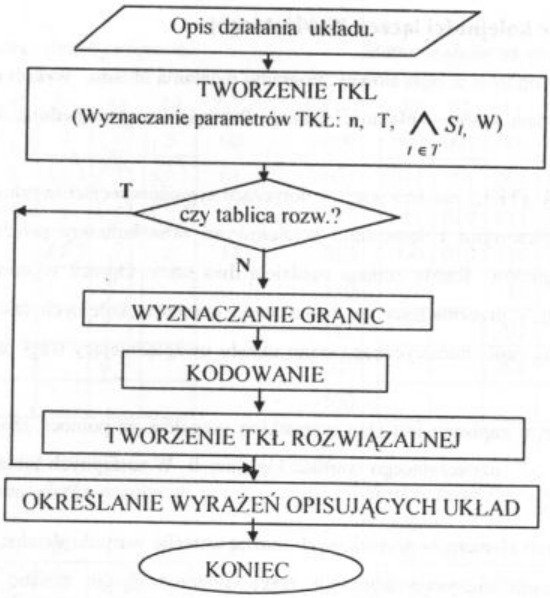
Tablica zawierająca takty sprzeczne logicznie jest tablicą nierozwiązalną. Wprowadzenie *elementów dodatkowych* do nierozwiązalnej TKŁ pozwala określić rozwiązalną TKŁ.

Dwa *stany* S występujące w taktach *i* oraz *j* są *sprzeczne logicznie*, jeżeli jeden z nich należy do warunków działania a drugi do warunków niedziałania (należą do warunków sprzecznych) przynajmniej jednego elementu wyjściowego lub dodatkowego. Parę taktów odpowiadających tym stanom nazywamy *taktami sprzecznymi* (*powtórzeniami sprzecznymi*).

Dwa *stany* S występujące w taktach *i* oraz *j* są *niesprzeczne logicznie*, jeżeli należą do niesprzecznych, czyli takich samych warunków każdego elementu wyjściowego i dodatkowego. Parę taktów odpowiadających tym stanom nazywamy *taktami niesprzecznymi* (*powtórzeniami niesprzecznymi*).

W przypadku, gdy TKŁ zawiera powtórzenia sprzeczne, warunki działania dla wyjść projektowanego układu nie mogą być jednoznacznie określone stanem wyłącznie elementów ujętych w tej tablicy.
Potrzebne są wtedy dodatkowe elementy, które wyeliminują sprzeczności istniejące w TKŁ.
Algorytm [1], [2] określania rozwiązania[1] metodą TKŁ można wyrazić schematem blokowym (rys. 9-30).

[1] Algorytm ten jest uwzględniony w programach zawartych w pilkach: Tkl-nau.zip Tkl-dos.zip Tkl-synt.zip dostępnych na witrynie internetowej: http://zmitac.iinf.polsl.gliwice.pl.



Rys. 9-30

W opisie algorytmu występują oznaczenia:

Z_{sp} - zbiór par stanów sprzecznych logicznie dla zadanej TKŁ,

Z_{nsp} - zbiór par stanów niesprzecznych logicznie,

n - liczba taktów w cyklu pracy układu,

T - zbiór numerów taktów tablicy (T = {1, 2, ... n}),

g_{max} - *granica* najbardziej oddalona od taktu pierwszego, aktualizowana podczas realizacji algorytmu,

$\{g_{min}\}$ - zbiór *granic* wyznaczony dla pierwszej pary stanów sprzecznych,

g_{min} - *granica* ze zbioru $\{g_{min}\}$ najmniej oddalona od taktu pierwszego,

W - zbiór elementów wyjściowych w,

Σ_w - zbiór wszystkich numerycznych stanów układu z taktów należących do warunków działania elementu wyjściowego w,

Π_w - zbiór wszystkich numerycznych stanów układu z taktów należących do warunków niedziałania elementu wyjściowego w,

S_i, S_j - para stanów S niesprzecznych w stosunku do siebie (posiadają wspólny dolny indeks w TKŁ),

S_i, Sj - para stanów S sprzecznych w stosunku do siebie (różnią się dolnym indeksem w TKŁ),

W wyniku realizacji bloku operacyjnego WYZNACZANIE GRANIC wyznaczone zostają miejsca zmian elementów dodatkowych zwane *granicami*. Dzielą one cykl pracy układu na części. *Granica* w nierozwiązalnej TKŁ rozdziela stany sprzeczne i wyznacza miejsce zmiany dodatkowego elementu w rozwiązalnej TKŁ. W opisywanym algorytmie [1], [2] *granica* wyrażona jest numerem taktu, po którym bezpośrednio występuje.

Ten blok stanowi kluczową część algorytmu. Można w nim wyróżnić cztery etapy:

Etap I. Określić zbiory: Z_{sp} i Z_{nsp}.

Etap II. Rozdzielić granicą stany sprzeczne w zakresie taktów tablicy od *1* do *n* uwzględniając: W1, W2, W3.

Etap III. Biorąc pod uwagę zamknięty cykl pracy układu należy zbadać tę część TKŁ, która jest zawarta między granicą najmniej odległą od ostatniego taktu (g_{max}) a granicą najmniej odległą od pierwszego taktu (g_{min}) i rozdzielić znalezione w tej części powtórzenia sprzeczne.

Etap IV. Ponieważ granica znaleziona w etapie III może zastąpić granicę jedną lub kilka ze zbioru $\{g_{min}\}$, to należy usunąć zbędne granice.

Opis wskazówek: [1], [2]

W1: Część cyklu nie może zawierać powtórzeń sprzecznych.

W2: Granica między częścią poprzednią α i następną β nie może przebiegać bezpośrednio po stanie *S*, który jest niesprzeczny logicznie w stosunku do stanu *S* kończącego część β lub należącego do części α. *Wyjątkiem* od tej zasady jest sytuacja, kiedy część α podzielona jest na fragmenty rozdzielone innymi częściami i bezpośrednio po każdym z niesprzecznych względem siebie stanów *S* w różnych fragmentach części α, wyznaczona jest granica poprzedzająca bezpośrednio część β.

W3: Granica między częścią poprzednią α i następną β nie może przebiegać bezpośrednio po stanie *S*, który jest sprzeczny logicznie w stosunku do stanu *S* należącego do części β.

Blok KODOWANIE dotyczy kodowania części wyznaczonych granicami stanem elementów dodatkowych z uwzględnieniem wskazówki W4.

W4: Części sąsiednie w czasie powinny być zakodowane sąsiednimi logicznie stanami elementów dodatkowych. Do kodowania należy użyć minimalnej liczby elementów dodatkowych.

W bloku TWORZENIE TKŁ ROZWIĄZALNEJ w oparciu o wyznaczone granice i zakodowane części cyklu tworzona jest rozwiązalna TKŁ. W stosunku do tablicy nierozwiązalnej TKŁ zawiera ona dodatkowe wiersze dotyczące elementów dodatkowych oraz dodatkowe takty umieszczone w miejscu granic między częściami. Dodatkowe takty i wiersze przeznaczone są dla zaznaczenia zmian elementów dodatkowych zgodnie z zakodowaniem części.

W bloku OKREŚLANIE WYRAŻEŃ OPISUJĄCYCH UKŁAD w oparciu o rozwiązalną TKŁ wyznacza się warunki działania i warunki niedziałania elementów *W* wyjściowych i dodatkowych. Zależnie od rodzaju elementów wybranych do realizacji układu, określa się:

- wyrażenia dla wyjść i elementów dodatkowych,
- lub funkcje wzbudzeń przerzutników użytych do ich realizacji.

Określanie wyrażeń dla elementu *W*

Zbiór *składników jedynki* elementu *W* wyjściowego lub dodatkowego tworzą wszystkie stany i tylko stany z taktów należących do warunków działania tego elementu, a zbiór *czynników zera* elementu *W* tworzą wszystkie stany i tylko stany z taktów należących do warunków niedziałania tego elementu. Stany nie występujące w tablicy należą do zbioru stanów Φ.

Określanie funkcji wzbudzeń przerzutników *sr* użytych do realizacji elementu *W*

Sposoby określania tych funkcji są szerzej opisane w rozdziale 9.5. W tym miejscu podany jest tylko sposób oparty bezpośrednio o rozwiązalną TKŁ:

1) takty rozpoczynające warunki działania (poprzedzające bezpośrednio zmianę na „+") elementu *W* tworzą zbiór składników jedynki wejścia *s*,
2) takty rozpoczynające warunki niedziałania (poprzedzające bezpośrednio zmianę na „-") elementu *W* tworzą zbiór składników jedynki wejścia *r*,
3) zbiór czynników zera wejścia *s* tworzą takty należące do warunków niedziałania elementu *W*,
4) zbiór czynników zera wejścia *r* tworzą takty należące do warunków działania tego elementu.

Dane wynikające z punktów 1 i 2 są wystarczające w przypadku realizacji *s* i *r* w postaci kanonicznej sumy. Dla realizacji *s* i *r* w postaci minimalnej potrzebne są również dane z punktów 3 i 4 albo dane dla elementu *W*.

Przykład 9.4

Cykl pracy układu

Takty			1	2	3	4	5	6	7	8	9	10	11	12	1	...
Stan elementów	x_1	2^0	−	+		−				+		−				itd.
	x_2	2^1	−				+	−								
	x_3	2^2	−										+	−		
	w	2^3	−		+				−		+				−	
Stan układu			0_1	1_1	9_1	8_1	10	8_2	0_1	1_1	9_1	8_1	12	8_2	0	...
Części				I		II		III				IV		I		
				β		α		β				α		β		
Stan q				−		+		−				+		−		

Rys. 9-31

Rysunek 9-31 podaje opis pracy układu w postaci tablicy kolejności łączeń. Stany sprzeczne logicznie na tym rysunku różnią się dolnymi indeksami, a stany niesprzeczne logicznie posiadają wspólny dolny indeks. Z tablicy wynika, że w układzie występują:

- pary sprzecznych stanów $(8)_{w x_3x_2x_1}$ w taktach: (*4, 6*), (*4, 12*), (*6, 10*), (*10, 12*),
- pary niesprzecznych stanów $(8)_{w x_3x_2x_1}$ w taktach: (4, 10), (6, 12),
- pary niesprzecznych stanów $(0)_{w x_3x_2x_1}$ w taktach: (1, 7).

Ze względu na W1 należy rozdzielić *granicą* pary stanów sprzecznych *Granicę* między taktami nr 4 i 6 nie można umieścić bezpośrednio po takcie 4 ze względu na W3 i musi być wyznaczona po takcie 5 nie należącym do powtórzeń. Podobnie *granicę* między taktami 10 i 12 umieszczono po takcie 11.

Granica między taktami 6 i 10 nie może przebiegać bezpośrednio po takcie 6 (ze względu na W3) i może przebiegać po takcie 7. 8 lub 9. Umieszczono ją po takcie 9. *Granica* między taktem 11 i występującym po nim (po powtórzeniu cyklu pracy układu) taktem 4 może przebiegać po taktach: 1. 2 lub 3. Wybrano ją po takcie 3. W ten sposób wyznaczono cztery *granice* po taktach: 3. 5. 9. 11 dzielące cykl pracy układu na cztery części: I. II. III i IV.

Części II i IV można połączyć w jedną część α, ponieważ nie zawiera ona powtórzeń sprzecznych (W1) a granice kończące oba fragmenty części α występują po stanach nie należących do powtórzeń (więc nie ma problemu opisanego w W2 i W3). Połączenie części I i III (nie zawierających łącznie powtórzeń sprzecznych) w jedną część β jest możliwe dzięki temu. że granice kończące obydwa fragmenty tak utworzonej części β stanowią *wyjątek* opisany we wskazówce W2.

Gdyby przykładowo zamiast po takcie 3 wybrać granicę po stanie 1 w takcie 2, wtedy połączenie części I i III nie byłoby możliwe ze względu na W2 (w części β w takcie 8 występowałby stan 1 będący niesprzecznym powtórzeniem stanu 1 z taktu 3 również należącego do części β). Dla spełnienia W2 należałoby granicę po takcie 9 zastąpić granicą po takcie 8. Podobnie, gdyby zamiast po takcie 3 wybrać granicę po takcie 1, to dla spełnienia W2 w części β, należałoby granicę po takcie 9 zastąpić granicą po takcie 7.

Części α i β wystarczy zakodować jednym elementem dodatkowym np.: *q* tak, jak podaje rys. 9-31.

Rozwiązalna TKŁ podana jest na rysunku 9-32. Zawiera ona tylko powtórzenia niesprzeczne.

Takty		1	2	3	3'	4	5	5'	6	7	8	9	9'	10	11	11'	12	1	...
Stan elementów	x_1 2^0	–	+			–					+			–					itd.
	x_2 2^1	–					+		–										
	x_3 2^2	–													+		–		
	w 2^3	–		+						–		+						–	
	q 2^4	–			+			–					+			–			
Stan układu		0	1	9	25	24	26	10	8	0	1	9	25	24	28	12	8	0	...

Cykl pracy układu

Rys. 9-32

Z warunków działania (linia ciągła) i warunków niedziałania (linia przerywana) wynika kanoniczna postać wyrażeń dla wyjścia *W* i elementu dodatkowego *Q*:

$$\begin{cases} W = \Sigma(1, 9, 25, 24, 26, 10, 28, 12)_{q\,w\,x_3x_2x_1} \\ W = \Pi(0, 8)_{q\,w\,x_3x_2x_1} \end{cases} \qquad \begin{cases} Q = \Sigma(9, 25, 24)_{q\,w\,x_3x_2x_1} \\ Q = \Pi(0, 1, 26, 10, 8, 28, 12)_{q\,w\,x_3x_2x_1} \end{cases}$$

Bezpośrednio z tablicy można też odczytać kanoniczną postać funkcji wzbudzeń przerzutników sr. Warto jednak sprawdzić, czy każdy z sygnałów wyjściowych zależy od swojego stanu poprzedniego.

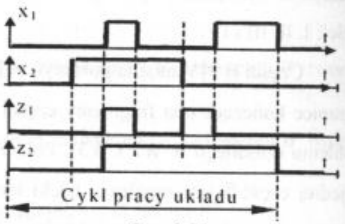
Jeżeli funkcja pewnego wyjścia jest kombinacyjna, to nie ma potrzeby realizować jej na przerzutniku. Tak jest w przypadku wyjścia W rozważanego przykładu. Można wykazać, że: $W = q + x_1 + x_2 + x_3$.

Dla elementu dodatkowego q określamy:

dla Q: $\begin{cases} s = \Sigma(9)_{q\,w\,x_3x_2x_1} \\ s = \Pi(0, 1, 26, 10, 8, 28, 12)_{q\,w\,x_3x_2x_1} \end{cases}$ $\begin{cases} r = \Sigma(26, 28)_{q\,w\,x_3x_2x_1} \\ r = \Pi(9, 25, 24)_{q\,w\,x_3x_2x_1} \end{cases}$

Przykład 9.5

Działanie układu o wejściach: x_1, x_2 i wyjściach z_1 i z_2 opisane jest wykresem czasowym rys. 9-33.



Rys. 9-33

Z niego wynika tablica kolejności łączeń (rys. 9-34). Zawiera ona sprzeczne powtórzenia stanów w cyklu pracy układu, więc nie jest tablicą rozwiązalną.

Na rysunku 9-34 linią ciągłą zaznaczono takty należące do warunków działania i linią przerywaną takty należące do warunków niedziałania. Wskazują one, że występują sprzeczne powtórzenia stanu 8 w taktach: 7 i 11. Stany sprzeczne rozróżniono dolnymi indeksami.

Takty		0	1	2	3	4	5	6	7	8	9	10	11	0
x_1	2^0	-		+			-				+		-	
x_2	2^1	-	+						-					
z_1	2^2	←-	----→	←-	+	----→	←-	----→	←-	+	←-	----→	←-	----→
z_2	2^3	←-	----→	----→	←-	+							←-	-
NSU		0	2	3	7	15	14	10	8_1	12	13	9	8_2	0

Cykl pracy układu

Rys. 9-34

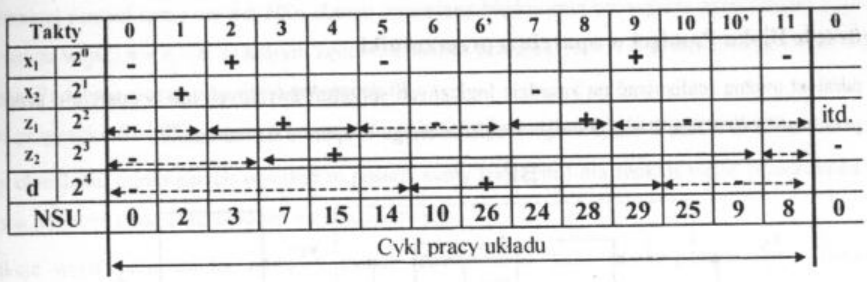
W tablicy na rys 9-35 wskazano podział cyklu pracy układu na części nie zawierające powtórzeń sprzecznych i uwzględniające warunki W1 - W3.

Takty		0	1	2	3	4	5	6	7	8	9	10	11	0
x_1	2^0	-		+			-				+		-	
x_2	2^1	-	+						-					
z_1	2^2	←-	----→	←-	+	----→	←-	----→	←-	+	←-	----→	←-	----→	itd.
z_2	2^3	←-	----→	----→	←-	+							←-	-	
NSU		0	2	3	7	15	14	10	8_1	12	13	9	8_2	0	

Cykl pracy układu

Rys. 9-35

Po wprowadzeniu elementu dodatkowego do TKŁ otrzymujemy rozwiązalną TKŁ (rys. 9-36).



Rys. 9-36

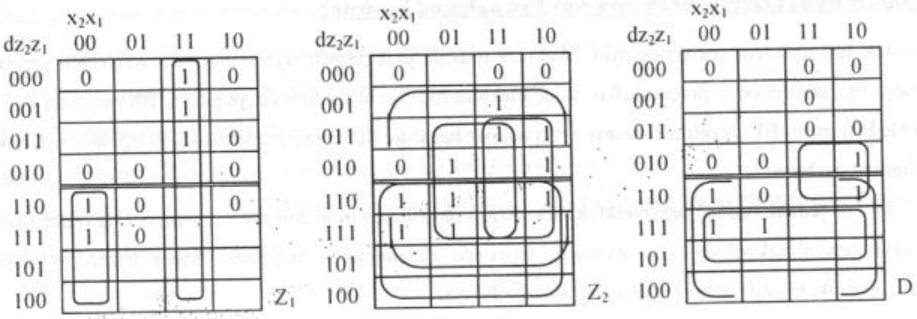
Z warunków działania i warunków niedziałania zaznaczonych na rys. 9-36 wynikają wyrażenia opisujące układ:

$$\begin{cases} Z_1 = \Sigma\,(3, 7, 15, 24, 28)_{d z_2 z_1 x_2 x_1} \\ Z_1 = \Pi\,(0, 2, 14, 10, 26, 29, 25, 9, 8)_{d z_2 z_1 x_2 x_1} \end{cases}$$

$$\begin{cases} Z_2 = \Sigma\,(7, 15, 14, 10, 26, 24, 28, 29, 25, 9)_{d z_2 z_1 x_2 x_1} \\ Z_2 = \Pi\,(0, 2, 3, 8)_{d z_2 z_1 x_2 x_1} \end{cases}$$

$$\begin{cases} D = \Sigma\,(10, 26, 24, 28, 29)_{d z_2 z_1 x_2 x_1} \\ D = \Pi\,(0, 2, 3, 7, 15, 14, 25, 9, 8)_{d\,z2\,z1\,x2\,x1} \end{cases}$$

Rysunek 9-37 przedstawia wartości tych funkcji w siatkach Karnaugh'a.



Rys. 9-37

Można z nich określić minimalną postać wyrażeń ogólnych dla wyjść: Z_1, Z_2 i elementu dodatkowego D:

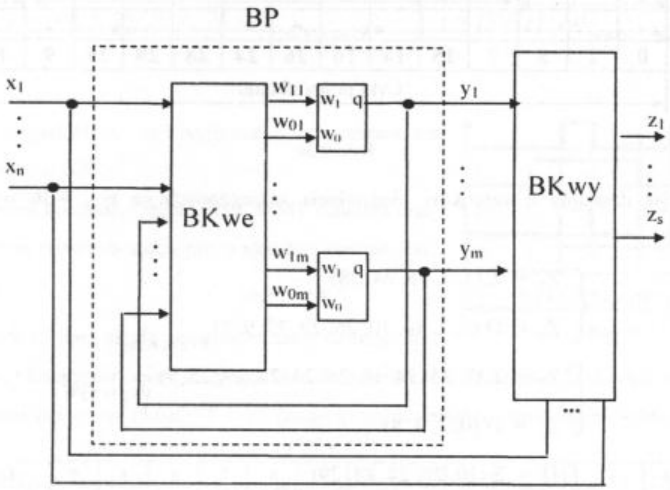
$Z_1 = x_2 x_1 + d\,\overline{x_2}\,\overline{x_1}$ - (zależność kombinacyjna).

$Z_2 = z_1 + d + z_2 x_1 + z_2 x_2$

$D = d z_1 + d\,\overline{x_1} + z_2 \overline{z_1} x_2$

9.5. Realizacja Bloku Pamięci w oparciu o przerzutniki

Elementy pamięci można realizować na zasadzie logicznych sprzężeń zwrotnych lub w oparciu o przerzutniki. Rys. 9-38 przedstawia strukturę Bloku Pamięci zrealizowanego w oparciu o przerzutniki.



Rys. 9-38

Funkcje wzbudzeń przerzutników, to funkcje realizujące sygnały wejściowe (w_1, w_0) przerzutników. Układ realizujący te funkcje nazywamy Blokiem Kombinacyjnym wejściowym (BKwe).

Czy funkcje wejść przerzutników powinny być wolne od hazardu?

Przerzutnik jest układem sekwencyjnym. Niektóre rodzaje przekłamań występujące na wejściach przerzutników mogą być utrwalone w przerzutniku. Więc niektóre rodzaje hazardu występujące w BKwe mogą być przyczyną błędnej pracy BP. Jeżeli nie znamy przepadków hazardu, które są nieszkodliwe, to ogólnie w BKwe należy eliminować hazard.

Można jednak podać przypadki, kiedy hazard określonego rodzaju nie ma wpływu na pracę przerzutnika, czyli jest nieszkodliwy. Oto wybrane, łatwe do zapamiętania przypadki, kiedy hazard występujący w realizacji funkcji wejść przerzutnika jest nieszkodliwy:

P1. Kiedy w przerzutniku *sr* obydwa wejścia: *s* i *r* realizowane są w postaci normalnej sumy, wtedy nie ma potrzeby eliminacji hazardu (*HSd*) dla tych wejść.

P2. Kiedy w przerzutniku $\bar{s}\ \bar{r}$ obydwa wejścia: $\bar{s}$ i $\bar{r}$ realizowane są w postaci normalnej iloczynu, wtedy nie ma potrzeby eliminacji hazardu (*HSn*) dla tych wejść.

Uzasadnienie przypadku P1. Chwilowe przekłamania typu: $1 \rightarrow 0 \rightarrow 1$ występujące na wejściu *s* (lub *r*) przerzutnika *sr* np. z powodu *HSd* nie mają wpływu na pracę tego przerzutnika, jeżeli jednocześnie na drugim jego wejściu *r* (lub *s*) jest wartość *0* bez przekłamań. Podczas rozważanego przejścia dla *s* (lub *r*) drugie wejście przerzutnika jest w stanie *0* i bez przekłamań typu *HSn*, bo zabroniony jest stan wejść: $sr = 11$

i w normalnej postaci sumy nie ma *HSn*. Zatem omawiany błędny stan na wejściu przerzutnika *sr* jest stanem nieaktywnym wejść: $s = r = 0$, w którym zgodnie z tablicą przejść (rys. 9-17) przerzutnik *sr* pamięta swój stan poprzedni. Podobnie można uzasadnić przypadek P2.

Opisane przypadki: P1 i P2 pozwalają na uproszczenie sposobu syntezy wejść przerzutników:

1. Przy określaniu minimalnych wyrażeń w postaci sumy (iloczynu) dla funkcji wejść przerzutnika *sr* ($\bar{s}\ \bar{r}$) nie ma potrzeby tworzenia wyrażeń antyhazardowych.
2. Funkcje wejść przerzutnika *sr* ($\bar{s}\bar{r}$) można realizować w kanonicznej postaci sumy (iloczynu) np. z zastosowaniem komutatorów lub pamięci stałej.
3. Koszt realizacji postaci kanonicznej w oparciu o układy programowane typu PLS lub PGA, jest porównywalny z kosztem realizacji postaci minimalnej. Zatem w przypadku realizacji BKwe z zastosowaniem tych układów, można zastosować prostszy algorytm określania kanonicznej postaci sumy (iloczynu) dla funkcji wejść przerzutnika *sr* ($\bar{s}\bar{r}$) w miejsce bardziej złożonego dającego w wyniku postać minimalną.

W przypadku realizacji BP na zasadzie sprzężeń zwrotnych również można podać przypadki kiedy hazard jest nieszkodliwy (opisane w przykładzie 15.1). Nie znając ich przy określaniu wyrażeń dla BP ogólnie należy eliminować hazard.

Można podać różne metody określania funkcji wzbudzeń przerzutników w tym następujące dla przerzutników o wejściach prostych:

1) *metoda oparta o tablice wzbudzeń przerzutników,*
2) *metoda porównywania wyrażeń logicznych* (opisujących BP i realizowanych przez przerzutnik),
3) *metoda bezpośrednia oparta o siatkę stanów wewnętrznych elementu pamięci:*
   a) *z rozwiązaniem w postaci minimalnej,*
   b) *z rozwiązaniem w postaci kanonicznej,*
4) *metoda bezpośrednia oparta o rozwiązalną tablicę kolejności łączeń.*

Dla przerzutników o wejściach zanegowanych funkcje wejść są przeciwne w stosunku do funkcji wejść przerzutników o wejściach prostych.

Metody wymienione w punktach: 1, 2, 3a są przydatne w przypadku realizacji BKwe w oparciu o bramki, a także w oparciu o układy programowane typu PLA.

Metody wymienione w punktach: 3b oraz 4 oparte są o algorytmy znacznie prostsze niż algorytmy pozostałych metod. Są one przydatne szczególnie w przypadku realizacji BKwe z zastosowaniem komutatorów, pamięci stałej jak również układów programowanych typu PLS lub PGA. Chociaż wyniki uzyskane tymi metodami nie zawierają pełnej informacji o wartościach funkcji wzbudzeń przerzutnika (brak informacji o stanach Φ), to jednak są one wystarczające w wymienionych przypadkach realizacji BKwe.

Metoda oparta o tablice wzbudzeń przerzutników

Z binarnej siatki stanów wewnętrznych w oparciu o tablice wzbudzeń przerzutnika określa się siatki dla wejść przerzutnika. Z siatek tych można określić wyrażenia dla funkcji wzbudzeń przerzutników.

Przykład 9.6

Na rys. 9-39 podano siatkę stanów jednego z dwóch elementów pamięci Y (rys. 9-39a).

a)

y x \ ab	00	01	11	10
00	0	1	Φ	1
01	0	0	Φ	0
11	Φ	0	Φ	1
10	0	1	Φ	1

Y

b)

y→Y	s r
0 0	0 Φ
0 1	1 0
1 0	0 1
1 1	Φ 0

c)

y x \ ab	00	01	11	10
00	0	1	Φ	1
01	0	0	Φ	0
11	Φ	0	Φ	Φ
10	0	Φ	Φ	Φ

s

d)

y x \ ab	00	01	11	10
00	Φ	0	Φ	0
01	Φ	Φ	Φ	Φ
11	Φ	1	Φ	0
10	1	0	Φ	0

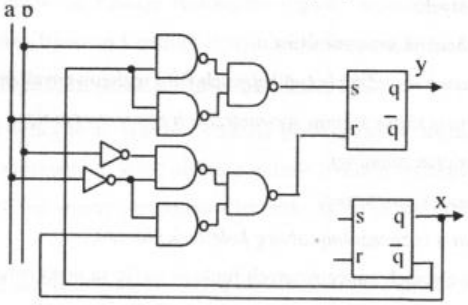
r

Rys. 9-39

Dla realizacji tego elementu wybrano przerzutnik *sr*, dla którego tablicę wzbudzeń podaje rys. 9-39b, a siatki funkcji wzbudzeń podają rysunki: 9-39c i 9-39d. Z nich wynikają wyrażenia minimalne:

$$s = \bar{x} \cdot b + \bar{x} \cdot a \qquad r = \bar{a} \cdot \bar{b} + x \cdot \bar{a}$$

Są one przydatne dla realizacji wejść: *s* i *r* w oparciu o bramki np. rys. 9-40.



Rys. 9-40

Metoda porównywania wyrażeń logicznych

Z binarnej siatki stanów wewnętrznych określa się wyrażenie podobne do wyrażenia realizowanego przez zastosowany przerzutnik (w postaci sumy dla przerzutników z dominującym wejściem wpisującym lub w postaci iloczynu dla przerzutników z dominującym wejściem zerującym).

Jeżeli wyrażenie zawiera kilka wyrazów (składników w postaci sumy lub czynników w postaci iloczynu) zależnych od stanu poprzedniego rozważanego przerzutnika *q*, to należy te wyrazy zgrupować względem argumentu *q*. Porównując otrzymane wyrażenie z wyrażeniem realizowanym przez przerzutnik, określa się funkcje wzbudzeń przerzutnika.

Uwaga: Wyniki uzyskane metodą porównywania wyrażeń lub opartą o tablice wzbudzeń nie zapewniają tego, że pomocnicze wyjście przerzutnika jest negacją jego wyjścia głównego. Chcąc w takim przypadku korzystać z wyjścia pomocniczego q' jako negacji, trzeba dokonać sprawdzenia, czy jest to dozwolone.

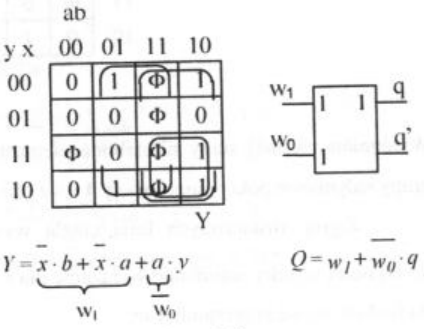
Sprawdzenie to w przypadku przerzutników klasy A, polega na utworzeniu funkcji iloczynu: $w_1 \cdot w_0$ i sprawdzeniu, czy w stanach pracy układu funkcja ta jest równa „0". Jeżeli tak, można korzystać z q' jako negacji q, a także można zastąpić przerzutnik klasy A przerzutnikiem *sr*.

W przypadku przerzutników klasy B, utworzy się funkcję sumy: $w_1 + w_0$ i sprawdza się, czy w stanach pracy układu funkcja ta jest równa „1". Jeżeli tak, można korzystać z q' jako negacji q, a także można zastąpić przerzutnik klasy B przerzutnikiem $\bar{s}\bar{r}$.

Przykład 9.7

Dla elementu pamięci Y z przykładu 9.6 poszukiwane są wyrażenia: w_1, w_0 dla przerzutnika klasy A z dominującym wejściem wpisującym. Na rys. 9-41 podana jest siatka stanów wewnętrznych oraz wyrażenie w postaci sumy dla elementu pamięci: Y oraz wyrażenie dla Q rozważanego przerzutnika. Z porównania wyrażeń: dla Y i dla Q przerzutnika wynikają funkcje wzbudzeń przerzutnika :

$$w_1 = \bar{x} \cdot b + \bar{x} \cdot a, \quad w_0 = \bar{a}$$



Rys. 9-41

Chcąc korzystać z wyjścia pomocniczego q' przerzutnika jako negacji jego wyjścia głównego q, trzeba dokonać następującego sprawdzenia. Określany jest iloczyn:

$$w_1 \cdot w_0 = \left(\bar{x} \cdot b + \bar{x} \cdot a\right) \bar{a} = \bar{x} \cdot b \cdot \bar{a}$$

Iloczyn ten przyjmuje wartość „1" w stanach pracy układu, więc nie można korzystać z q' jako negacji q i nie można zastąpić przerzutnika wybranego do realizacji przerzutnikiem *sr*.

Metoda bezpośrednia oparta o siatkę stanów elementu pamięci – z rozwiązaniem w postaci minimalnej

1. W siatce stanów elementu pamięci Y, zaznaczyć (np. przez pogrubienie) stany niestabilne (**1** i **0**).
2. Wyróżnione stany **1** połączyć w grupy jedynkowe według zasad obowiązujących przy minimalizacji wyrażeń. Dla powiększenia grupy można wykorzystać stany ϕ i nie wyróżnione stany 1. Utworzone grupy wskazują stan aktywny (równy *1*) wejścia *s*, czyli są grupami jedynkowymi dla *s*.
3. Suma wyrażeń opisujących grupy zaznaczone w punkcie 2 określa wyrażenie dla *s* przerzutnika *sr*.
4. Wyróżnione stany **0** połączyć w grupy według zasad obowiązujących przy minimalizacji wyrażeń. Dla powiększenia grupy można wykorzystać stany ϕ oraz nie wyróżnione stany 0. Utworzone grupy wskazują stan aktywny (równy *1*) wejścia *r*, czyli są grupami jedynkowymi dla *r*.

5. Suma wyrażeń opisujących grupy zaznaczone w punkcie 4 (pamiętając, że są to grupy jedynkowe dla wejścia r!) określa wyrażenie dla r przerzutnika sr.

Uwaga: W punktach 2 i 4 nie trzeba tworzyć grup antyhazardowych.

Przykład 9.8

Na rysunku 9-42 a) podana jest siatka stanów elementu pamięci Y.

a)

y \ ab	00	01	11	10
00	0	**1**	Φ	**1**
01	0	0	Φ	0
11	Φ	**0**	Φ	1
10	**0**	1	Φ	1

Y

b)

y \ ab	00	01	11	10
00	0	**1**	Φ	**1**
01	0	0	Φ	0
11	Φ	**0**	Φ	1
10	**0**	1	Φ	1

Y

Rys. 9-42

Wyróżniono w niej stany niestabilne elementu Y przez pogrubienie. W siatce b) zaznaczono linią ciągłą grupy jedynkowe pokrywające stany **1** oraz linią przerywaną grupy zerowe pokrywające stany **0**.

Z grup oznaczonych linią ciągłą wynika minimalne wyrażenie dla s. Z grup oznaczonych linią przerywaną wynika minimalne wyrażenie dla r. Funkcje wejść dla przerzutnika $\bar{s}\ \bar{r}$ są przeciwne w stosunku do funkcji wejść przerzutnika sr:

$$s = \bar{x} \cdot b + \bar{x} \cdot a, \qquad r = \bar{b} \cdot \bar{a} + \bar{a} \cdot x, \qquad \bar{s} = \overline{\bar{x} \cdot b + \bar{x} \cdot a}, \qquad \bar{r} = \overline{\bar{b} \cdot \bar{a} + \bar{a} \cdot x}.$$

Warto zwrócić uwagę na zależność między grupami zaznaczonymi na rys. 9-39 c i d a grupami zaznaczonymi na rys. 9-41.

Metoda bezpośrednia oparta o siatkę stanów wewnętrznych elementu pamięci z rozwiązaniem w postaci kanonicznej

1. W siatce stanów elementu pamięci Y, zaznaczyć (np. przez pogrubienie) stany niestabilne (**1** i **0**).
2. Wyróżnione stany **1** wskazują stany aktywne (równe „1") wejścia s, czyli stany, w których $s = 1$.
3. Suma wyrażeń opisujących wyróżnione stany **1** zaznaczone w punkcie 2 określa wyrażenie dla s w kanonicznej postaci.
4. Wyróżnione stany **0** wskazują stany aktywne (równe „1") wejścia r, czyli stany w których $r = 1$.
5. Suma wyrażeń opisujących wyróżnione stany **0** zaznaczone w punkcie 4 określa wyrażenie dla r w kanonicznej postaci. Stany **0** z punktu 4 są zerowymi dla Y ale jedynkowymi dla r, więc przy tworzeniu wyrażenia dla r należy je traktować jako składniki jedynki dla r.

Przykład 9.9

Rysunek 9-43 przedstawia siatkę stanów elementu pamięci Y. Określenie funkcji wejść *s*, *r* w kanonicznej postaci sumy może nastąpić bezpośrednio z siatki stanów elementu Y. Zaznaczono w niej stany niestabilne przez pogrubienie.

y x \ ab	00	01	11	10
00	0	**1**	Φ	**1**
01	0	0	Φ	0
11	Φ	**0**	Φ	1
10	**0**	1	Φ	1

Y

Rys. 9-43

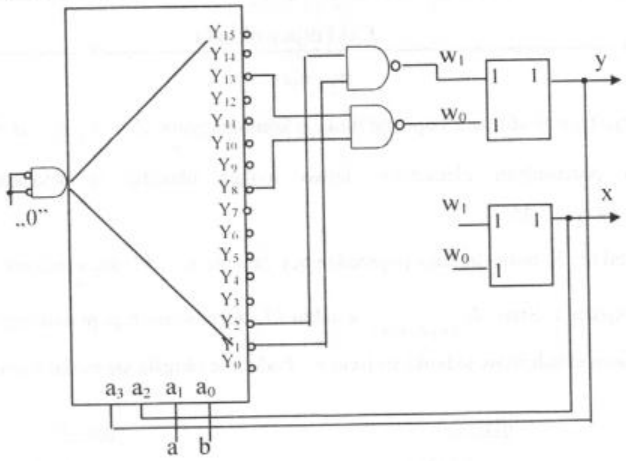
Z wyróżnionych stanów **1** wynika kanoniczna postać wyrażenia dla *s*:

$$s = \bar{y}\cdot\bar{x}\cdot\bar{a}\cdot b + \bar{y}\cdot\bar{x}\cdot a\cdot\bar{b} \qquad s = \sum(1,2)_{yxab}$$

Z wyróżnionych stanów **0** wynika kanoniczna postać wyrażenia dla *r*:

$$r = y\cdot\bar{x}\cdot\bar{a}\cdot\bar{b} + y\cdot x\cdot\bar{a}\cdot b \qquad r = \sum(8,13)_{yxab}$$

Postać kanoniczna jest przydatna przy realizacji *s* i *r* np. w oparciu o demultiplekser (rys. 9-44).



Rys. 9.9-44

Metoda bezpośrednia oparta o tablicę kolejności łączeń

1. W rozwiązalnej TKŁ dla elementu *q* (wyjściowego lub dodatkowego), który ma być zrealizowany w oparciu o przerz. ***sr*** poszukuje się taktów bezpośrednio poprzedzających zmiany tego elementu na „**+**".

2. Stany numeryczne układu z taktów znalezionych w punkcie 1 należą do zbioru składników jedynki funkcji wejścia *s* przerzutnika *sr* (do zbioru czynników zera funkcji wejścia *s* należą stany określone jako warunki niedziałania elementu *q*).
3. Dla elementu *q*, poszukuje się taktów bezpośrednio poprzedzających zmiany tego elementu na „-".
4. Stany numeryczne układu z taktów znalezionych w punkcie 3 należą do zbioru składników jedynki funkcji wejścia *r* przerzutnika *sr* (do zbioru czynników zera funkcji wejścia *r* należą stany określone jako warunki działania elementu q).

Przykład 9.10

Działanie układu o wejściach: x_1, x_2, wyjściach: Z_1, Z_2 i elemencie dodatkowym D opisuje (rys. 9-45) rozwiązalna tablica kolejności łączeń (TKŁ).

Rys. 9-45

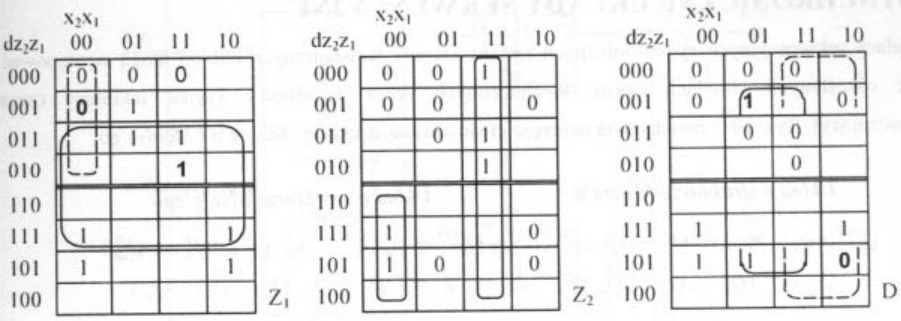
Warto zauważyć (rys. 9-46), że Z_2 opisuje funkcja kombinacyjna: $Z_2 = x_2 x_1 + d \overline{x_2}\ \overline{x_1}$. Funkcje wejść przerzutnika *sr* dla pozostałych elementów łatwo można określić w kanonicznej postaci sumy bezpośrednio z tablicy na rys. 9-45.

Stan $(11)_{d z_2 z_1 x_2 x_1}$ z taktu: 3, bezpośrednio poprzedzający zmianę na „+" dla elementu Z_1 należy do zbioru składników jedynki wejścia *s*. Stan $(4)_{d z_2 z_1 x_2 x_1}$ z taktu *11*, bezpośrednio poprzedzający zmianę na „-" tego elementu należy do zbioru składników jedynki wejścia *r*. Podobnie określa się funkcje *sr* dla elementu *d*.

Oto wyniki:

dla Z_1	dla D
$s = \Sigma(11)_{d z_2 z_1 x_2 x_1}$	$s = \Sigma(5)_{d z_2 z_1 x_2 x_1}$
$r = \Sigma(4)_{d z_2 z_1 x_2 x_1}$	$r = \Sigma(22)_{d z_2 z_1 x_2 x_1}$

Funkcje wejść przerzutników *sr* w minimalnej postaci, można określić bezpośrednio w oparciu o siatki stanów (rys. 9-46).



Rys. 9-46

Siatki te wypełnione są w oparciu o warunki działania i niedziałania zaznaczone na rys. 9-45 dla elementów: wyjściowych Z_1 i Z_2 i dodatkowego D. Stany niestabilne elementów wyróżnione w siatkach przez pogrubienie odpowiadają podanym wcześniej stanom w kanonicznej postaci sumy wejść *s*, *r* określonym bezpośrednio z rozwiązalnej TKŁ. Oto wyniki minimalnych wyrażeń otrzymane z siatek na rys. 9-46:

$Z_2 = x_2 x_1 + d \overline{x_2}\ \overline{x_1}$ - funkcja kombinacyjna (nie warto jej realizować na przerzutniku).

dla Z_1: $s = z_2$ $r = \overline{d}\ \overline{x_2}\ \overline{x_1}$

dla D: $s = z_1 \overline{z_2}\ x_1$ $r = x_2 \overline{z_2}$.

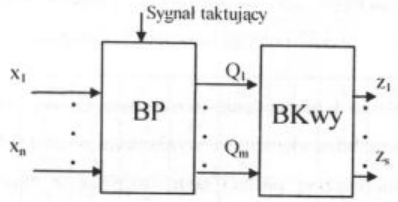
10. SYNCHRONICZNE UKŁADY SEKWENCYJNE

W układach sekwencyjnych synchronicznych zmiany stanów wewnętrznych układu mogą występować tylko w ściśle określonych chwilach czasu, wyznaczonych przez dodatkowy sygnał taktujący (zegarowy, synchronizujący). Rys. 10-1 przedstawia najczęściej stosowane struktury: *Moore'a* i *Mealye'go*.

Układ o strukturze Moore'a

$$Q_k^{n+1} = \varphi_k(x_1^n, \ldots, x_n^n, Q_1^n, \ldots, Q_k^n, \ldots, Q_m^n)$$

$$Z_j = f_j(Q_1, \ldots, Q_k, \ldots, Q_m)$$

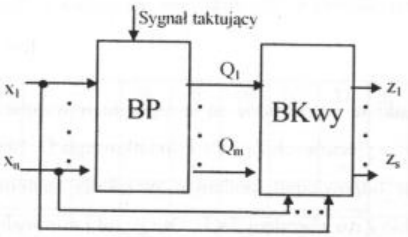


Aktualny stan wyjść automatu zależy tylko od aktualnego stanu wewnętrznego.

Układ o strukturze Mealy'ego

$$Q_k^{n+1} = \varphi_k(x_1^n, \ldots, x_n^n, Q_1^n, \ldots, Q_k^n, \ldots, Q_m^n)$$

$$Z_j = f_j(x_1, \ldots, x_n, Q_1, \ldots, Q_k, \ldots, Q_m)$$



Aktualny stan wyjść zależy od aktualnego stanu wejść i aktualnego stanu wewnętrznego

Rys. 10-1

Sygnał taktujący (zegarowy) oddziałuje w ten sposób na BP, że stan układu określony sygnałami: $Q_1, \ldots, Q_m$ może się zmieniać tylko w chwilach czasu wyznaczonych impulsami zegarowymi. W związku z tym w opisie funkcji przejść dla BP występują górne indeksy n i $n+1$ dotyczący przedziału czasu: n i $n+1$.

Przedziały czasu: n, $n+1$ wyznaczone są przez dwa kolejne aktywne zbocza sygnału zegarowego. Sygnały: $Q_1, \ldots, Q_m$ posiadają stałą wartość w przedziale czasu n określoną jako: $Q_1^n, \ldots, Q_m^n$.

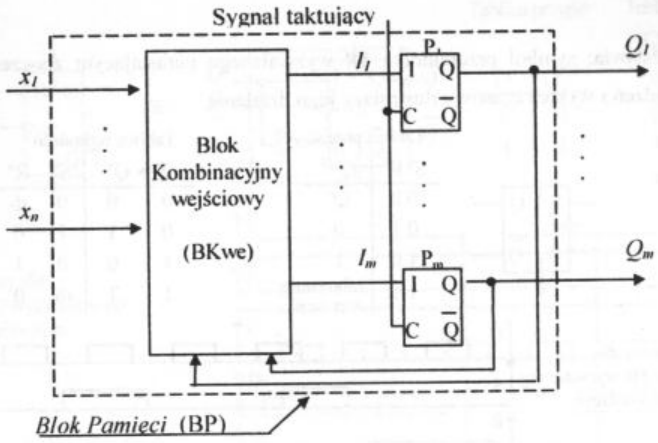
Sygnały: $x_1, \ldots, x_n$ mogą zmieniać się asynchronicznie w przedziale czasu n. Dla nich oznaczenia: $x_1^n, \ldots, x_n^n$, należy rozumieć jako wartości tych sygnałów takie, jakie są przez pewien czas (co najmniej czas wyprzedzenia) przed kolejnym aktywnym zboczem sygnału zegarowego warunkującego przejście do przedziału $n+1$.

Do realizacji Bloku Pamięci (BP) zazwyczaj stosuje się przerzutniki synchroniczne. Rysunek 10-2 przedstawia strukturę bloku pamięci z wyróżnieniem przerzutników synchronicznych.

Wtedy w Boku Pamięci można wyróżnić przerzutniki[1]: $P_1, \ldots, P_m$ oraz Blok Kombinacyjny wejściowy (BKwe) realizujący funkcje wejść informacyjnych przerzutników: $I_1, \ldots, I_m$. Dla k-tego wejścia informacyjnego realizowana jest funkcja kombinacyjna: $Ik = \Psi_k(x_1, \ldots, x_n, Q_1, \ldots Q_m)$.

W opisie bloków kombinacyjnych: BKwy i Bkwe pomija się górny indeks dotyczący przedziału czasu.

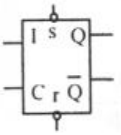
[1] Tematyka przerzutników jest uwzględniona w programie: Prze-nau.zip dostępnym na witrynie internetowej: http://zmitac.iinf.polsl.gliwice.pl



Rys. 10-2

10.1. Przerzutniki synchroniczne

Rys. 10-3 przedstawia ogólny symbol i oznaczenia. Wejścia ustawiające dominują nad pozostałymi. Stan aktywny (najczęściej 0) wejścia wpisującego wymusza asynchronicznie stan 1 a stan aktywny wejścia zerującego wymusza stan 0 na wyjściu Q.



gdzie: *I* - wejście informacyjne (jedno lub kilka),
C - wejście zegarowe,
Q - wyjście,

wejścia ustawiające:
s - wpisujące (ang. set),
r - zerujące (ang. reset).

$$Q^{n+1} = f(Q^n, I^n).$$

Rys. 10-3

Przerzutniki synchroniczne, mogą mieć jedno lub kilka (zwykle nie więcej niż 2) wejść informacyjnych.

Praktyczne zastosowanie znalazły głównie dwa typy przerzutników z wieloma wejściami informacyjnymi: *SR* (rys. 10-4) oraz *JK* (rys. 10-5). Z jednym wejściem informacyjnym najczęściej stosowane są przerzutniki typu *D* (rys. 10-6) i typu *T* (rys. 10-7).

Zależnie od sposobu wyzwalania przerzutnika (opisanego dalej) stosowane są dodatkowe symbole obok wejścia zegarowego:

- wyzwalanie narastającym zboczem,

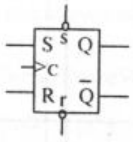
- wyzwalanie wysokim poziomem (C=1),

- wyzwalanie opadającym zboczem lub Master-Slave,

- wyzwalanie niskim poziomem (C=0).

Przerzutnik *SR*

Rysunek 10-4 przedstawia: symbol przerzutnika *SR* wyzwalanego narastającym zboczem a także: tablice przejść, tablice wzbudzeń i wykres czasowy ilustrujący jego działanie.



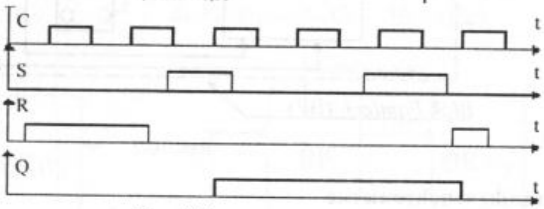
Tablica przejść

S^n R^n	Q^{n+1}
0 0	Q^n
0 1	0
1 0	1
1 1	zabroniony stan wejść

Tablica wzbudzeń

$Q^n \to Q^{n+1}$		S^n	R^n
0	0	0	Φ
0	1	1	0
1	0	0	1
1	1	Φ	0

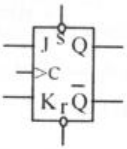
Wykres czasowy dla przerzutnika SR wyzwalaneg narastającym zboczem



Rys. 10-4

Przerzutnik *JK*

Rysunek 10-5 przedstawia: symbol przerzutnika *JK* wyzwalanego narastającym zboczem oraz: tablice przejść, tablice wzbudzeń i wykres czasowy ilustrujący jego działanie.



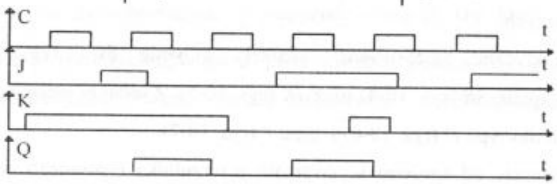
Tablica przejść

J^n K^n	Q^{n+1}
0 0	Q^n
0 1	0
1 0	1
1 1	$\overline{Q^n}$

Tablica wzbudzeń

$Q^n \to Q^{n+1}$		J^n	K^n
0	0	0	Φ
0	1	1	Φ
1	0	Φ	1
1	1	Φ	0

Wykres czasowy dla przerzutnika JK wyzwalanego narastającym zboczem

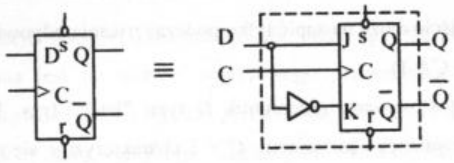


Rys. 10-5

Przerzutnik *D*

Wyjście Q w przedziale czasu $n+1$ (Q^{n+1}) przyjmuje wartość D^n, czyli taką, jaka jest na wejściu D (przez co najmniej czas wyprzedzenia) przed kolejnym impulsem zegarowym poprzedzającym przedział $n+1$.

Przerzutnik ten można zrealizować w oparciu o przerzutnik *JK* wprowadzając na wejście *K* sygnał przeciwny do sygnału wprowadzonego na wejście *J*. Rysunek 10-6 przedstawia: symbol przerzutnika *D* wyzwalanego narastającym zboczem, sposób jego realizacji w oparciu o przerzutnik *JK* a także: tablice przejść, tablice wzbudzeń i wykres czasowy ilustrujący jego działanie.



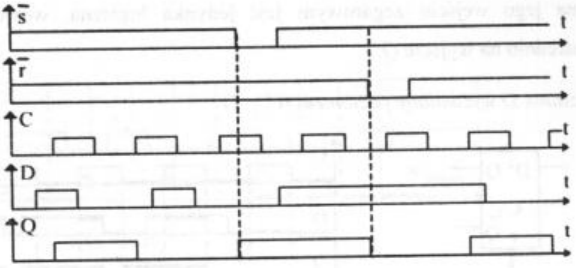
Tablica przejść

D^n	Q^{n+1}
0	0
1	1

Tablica wzbudzeń

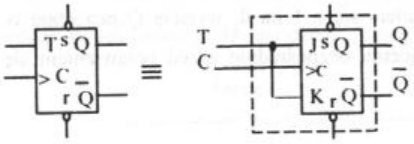
$Q^n \to Q^{n+1}$		D^n
0	0	0
0	1	1
1	0	0
1	1	1

Wykres czasowy dla przerzutnika D wyzwalanego narastającym zboczem



Rys. 10-6

Przerzutnik *T*



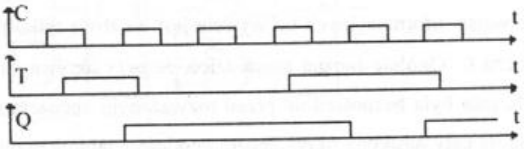
Tablica przejść

T^n	Q^{n+1}
0	Q^n
1	$\overline{Q^n}$

Tablica wzbudzeń

$Q^n \to Q^{n+1}$		T^n
0	0	0
0	1	1
1	0	1
1	1	0

Wykres czasowy dla przerzutnika T wyzwalanego narastającym zboczem



Rys. 10-7

Gdy w przedziale wartość $T^n = 0$, to w przedziale $n+1$ stan wyjścia Q nie ulega zmianie. Gdy wartość $T^n = 1$, to w przedziale $n+1$ stan wyjścia Q przyjmuje wartość przeciwną do tej, która była w przedziale n. Dla $T = 1$ każde pobudzenie wejścia zegarowego powoduje zmianę stanu wyjścia Q na przeciwną.

Sposoby wyzwalania przerzutników

Stan wyjścia w przerzutniku synchronicznym zmienia się w zależności od wejść informacyjnych w momentach określonych przez wejście zegarowe (taktujące). W tej części rozdziału w opisach przerzutników pominięte są wejścia ustawiające. Zwykle spotyka się trzy sposoby wyzwalania przerzutników synchronicznych:

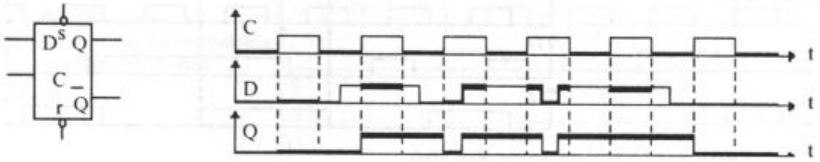
- poziomem,
- zboczem narastającym (przednim) lub opadającym (tylnym) – jednostopniowe,
- zboczem (narastającym i opadającym) – dwustopniowe typu Master-Slave.

Wyzwalanie poziomem

Oddziaływanie wejść informacyjnych na wyjście może nastąpić tylko podczas trwania odpowiedniego poziomu logicznego na wejściu taktującym ($C = 0$ lub $C = 1$).

W układach z pamięcią często jest stosowany przerzutnik D typu "latch" (rys. 10-8) wyzwalany poziomem. Działanie tego przerzutnika wyzwalanego poziomem: $C = 1$ charakteryzuje się tym, że w czasie, gdy na jego wejściu zegarowym jest jedynka logiczna, wówczas wejście informacyjne D oddziałuje bezpośrednio na wyjście Q.

Przerzutnik D wyzwalany poziomem (C = 1)



Rys. 10-8

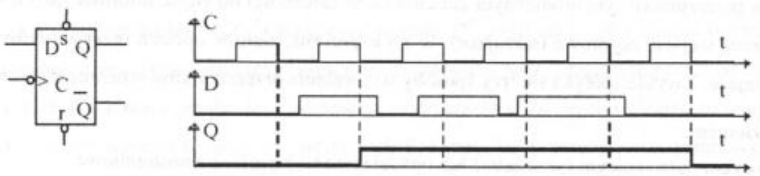
Wszystkie zmiany stanów wejścia D, zachodzące podczas $C = 1$ są asynchronicznie powtarzane na wyjściu Q (rys. 10-8). Kiedy stan wejścia zegarowego zmieni się z 1 na 0, wyjście Q pozostaje w stanie logicznym, odpowiadającym stanowi wejścia D występującemu bezpośrednio przed pojawieniem się zmiany z 1 na 0 sygnału zegarowego.

Wyzwalanie zboczem

Oddziaływanie wejść informacyjnych na wyjście jest możliwe tylko przy dodatnim ($C = 0 \rightarrow 1$) lub ujemnym ($C = 1 \rightarrow 0$) zboczu C. Ogólnie można powiedzieć, że przy aktywnym zboczu sprawdzana jest wartość wejść informacyjnych, jaka była bezpośrednio przed rozważanym zboczem i zależnie od tej wartości ustawiana jest wartość wyjścia na cały następny okres zegara zgodnie z tablicą przejść przerzutnika. Np. dla przerzutnika D (rys. 10-9) wyzwalanego opadającym (ujemnym, tylnym) zboczem wyjście Q na cały następny okres zegara przyjmuje wartość jaka była na wejściu D tuż przed rozważanym opadającym zboczem.

W katalogu elementów cyfrowych podana jest wymagana minimalna wartość czasu wyprzedzenia. Wartość tę należy rozumieć jako czas, w którym wejście informacyjne przerzutnika musi utrzymywać stałą wartość, aby wyjście Q^{n+1} mogło przyjąć żądaną wartość.

Przerzutnik D wyzwalany opadającym zboczem

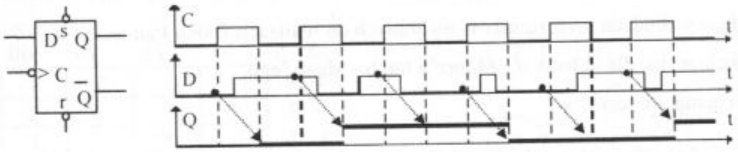


Rys. 10-9

Wyzwalanie dwustopniowe(Master-Slave)

Wpisanie informacji z wejść informacyjnych przebiega dwustopniowo. Podczas narastającego zbocza informacja wpisana jest do układu wejściowego przerzutnika (Master), a podczas opadającego zbocza informacja przekazywana jest na wyjście przerzutnika (Slave). Zależnie od budowy przerzutnika jego działanie może być takie, jak opisano wyżej (typ „idealny") lub nieco odmienne (typ „łapiący jedynkę").

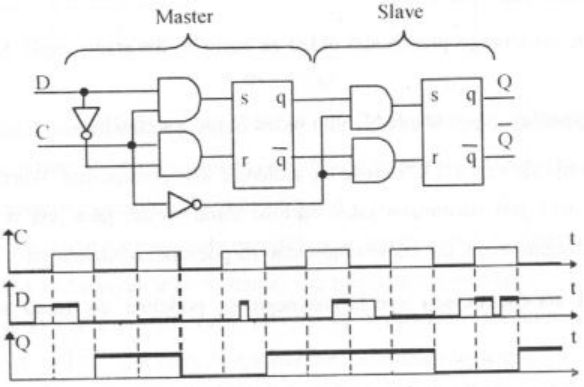
Przerzutnik D Master-Slave „idealny"



Rys. 10-10

Dla przerzutnika *D* typu: Master-Slave „idealnego" (rys. 10-10) wartość *D* odczytana przy narastającym zboczu C (wartość ta musi trwać pewien czas przed rozważanym zboczem i po tym zboczu) zapisana jest na wyjściu Q przy opadającym zboczu i pozostaje bez zmian przez cały następny okres zegara.

Przerzutnik D wyzwalany dwustopniowo (Master-Slave „łapiący jedynkę")



Rys. 10-11

Dla przerzutnika z rys. 10-11 wartość Q ustawiana jest na cały okres zegara zgodnie z zasadą:

- gdy dla $C = 1$ przez jakiś czas istnieje $D = 1$, to przy opadającym zboczu ustawiany jest stan $Q = 1$ (łapanie 1),
- gdy dla $C = 1$ istnieje cały czas $D = 0$, to przy opadającym zboczu ustawiana jest wartość $Q = 0$.

Możliwości zastosowania przerzutników są szerokie. Najważniejszą dziedziną zastosowań przerzutników są systemy zliczające, rejestry. Przerzutniki służą też do budowy sekwencyjnych układów sterowania.

10.2. Synteza synchronicznych układów sekwencyjnych metodą siatek przejść

Algorytm syntezy układów synchronicznych metodą siatek przejść [2]

1. Opisać program pracy układu: słownie, wykresem czasowym lub grafem przejść,
2. Określić siatkę przejść/wyjść układu,
3. Określić siatkę przejść/wyjść układu zredukowaną dla układu o strukturze Moore'a lub Mealy'ego,
4. Zakodować dwójkowo zredukowaną siatkę przejść/wyjść,
5. Określić dwójkową (binarną) siatkę przejść/wyjść,
6. Określić funkcje wzbudzeń przerzutników wybranych do realizacji Bloku Pamięci,
7. Określić funkcje wyjść dla układu: a - Moore'a lub b - Mealy'ego,
8. Narysować schemat logiczny.

Ad 1 Zasady opisu są takie same, jak dla układów asynchronicznych sekwencyjnych.

Ad 2. Siatka przejść/wyjść posiada kolumny odpowiadające stanom wejść i wiersze odpowiadające stanom układu. Po lewej stronie każdego wiersza zapisany jest odpowiadający mu stan S^n, czyli stan w przedziale czasu n przed aktualnie rozważanym impulsem zegarowym. Pierwotna siatka przejść/wyjść zazwyczaj jest siatką dla układu Moore'a. W niej, w osobnej kolumnie obok każdego wiersza jest zapisany stan wyjść, jaki odpowiada danemu stanowi układu. W kratce zawartej w wierszu dla stanu układu S^n i kolumnie dla stanu wejść X_i^n zapisany jest:

- stan S^{n+1} czyli stan, do którego przechodzi układ ze stanu S^n dla stanu wejść X_i, po kolejnym impulsie zegarowym,
- lub znak „-", w przypadku, gdy w stanie S^n, stan wejść X_i nie jest możliwy.

Stan wejść w n-tym przedziale czasu (X_i^n) może się zmieniać asynchronicznie. Wartość X_i^n, która warunkuje przejście do przedziału n+1 jest rozumiana jako wartość stanu wejść, jaka jest w czasie poprzedzającym aktywne zbocze sygnału zegarowego, tak jak to omówiono na początku rozdziału 10.

Ad 3. Zasady redukcji stanów układu synchronicznego są podobne do zasad redukcji stanów układu asynchronicznego.

Dwa wiersze można zredukować (dla struktury Moore'a lub Mealy'ego) wtedy i tylko wtedy, kiedy posiadają niesprzeczne przejścia, czyli w żadnej z kolumn rozważanych wierszy nie ma indeksów stanów sprzecznych. Zatem w kolumnach wierszy, które można zredukować występuje w nich jeden z przypadków: indeksy stanów zgodnych (lub zgodnych warunkowo i warunek jest spełniony), indeks i kreska lub dwie kreski.

Przez stany zgodne należy rozumieć stany (wiersze), które mają niesprzeczne stany wyjść i niesprzeczne przejścia. Dowolne dwa stany wewnętrzne są zgodne warunkowo (warunek: $j = k$), jeżeli są zgodne pod

[2] Komputerowa synteza tych układów z wizualizacją pośrednich wyników jest uwzględniona w programie: Synchr-s.zip dostępnym na witrynie internetowej: http://zmitac.iinf.polsl.gliwice.pl

warunkiem, że stany: j i k są zgodne. Jeżeli warunkiem zgodności stanów : j i k jest: $(j = k)$, to oczywiście ten warunek można pominąć przy redukcji tych stanów. Tak właśnie jest dla stanów 1 i 3 na rys. 10-12a.

Stany zgodne można zastąpić jednym stanem wewnętrznym. Odpowiadający mu wiersz otrzymuje się poprzez nałożenie na siebie wierszy przyporządkowanych rozważanym stanom zgodnym (rys. 10-12c).

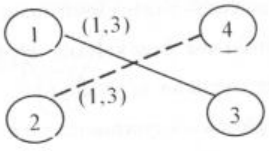
Jeżeli redukowane wiersze posiadają sprzeczny stan dla przynajmniej jednego wyjścia, to redukcja ich prowadzi do struktury Mealy'ego. Redukcja wierszy o niesprzecznych wyjściach może być wykorzystana zarówno dla struktury Moore'a jak i Mealy'ego.

a)

S^n	$x_1^n x_2^n$ 00	01	11	10	$Z_1 Z_2$
1	1	2	-	1	0 Φ
2	2	1	-	3	0 0
3	3	-	4	1	Φ 1
4	2	-	1	1	1 0
⋮					

S^{n+1}

b)



c)

S^n	$x_1^n x_2^n$ 00	01	11	10	$Z_1 Z_2$
1,3	1	2	4	1	0 1
2	2	1	-	3	0 0
4	2	-	1	1	1 0
⋮					

S^{n+1}

d)

Sn	$x_1^n x_2^n$ 00	01	11	10
1,3	1/01	2/00	4/10	1/01
2,4	2/00	1/01	1/01	3/01
⋮				

S^{n+1}/Z_1Z_2

Rys. 10-12

Stan wyjść w zredukowanej siatce przejść/wyjść: dla układu Moore'a zapisuje się w każdym wierszu w osobnej kolumnie. Dla układu Mealy'ego stan wyjść zapisuje się kratkach siatki, jako stan wyjść, który układ powinien przyjąć przy przejściach określonych tymi kratkami.

Ad 4. Kodowanie wierszy zredukowanej siatki przejść/wyjść jest dowolne, bez eliminacji wyścigu z użyciem minimalnej liczbę sygnałów kodujących jaka wystarczy dla rozróżnienia wierszy.

Ad 5. Dwójkową (binarną) siatkę przejść/wyjść określa się w oparciu o zredukowaną siatkę przejść/wyjść, wpisując w miejsce stanów: S^n i S^{n+1} ich kody dwójkowe (bez eliminacji wyścigu).

Ad 6. Istnieją rożne metody wyznaczenia funkcji wejść informacyjnych przerzutnika synchronicznego. Do najczęściej stosowanych należą:

- oparta o siatki stanów elementu pamięci i tablice wzbudzeń przerzutnika,
- oparta o transformację funkcji logicznej elementu pamięci,

} nie opisane w niniejszym rozdziale

- bezpośrednia, oparta wyłącznie o dwójkową siatkę stanów elementu pamięci metoda ta jest zilustrowana w przykładach 10.1, 10.2 i 10.3.

Ad 7. Funkcje wyjść dla układu Moore'a określa się na podstawie tablicy zależności wyjść od sygnałów stanu, która jest częścią składową binarnej siatki przejść/wyjść (każdy wiersz siatki jest zakodowany sygnałami stanów wewnętrznych i w każdym, jest zapisana wartość wyjść). Funkcje wyjść dla układu Mealy'ego określa

się na podstawie siatki zależności wyjść od sygnałów stanu i sygnałów wejściowych układu, która jest częścią składową binarnej siatki przejść/wyjść układu Mealy'ego (w każdej kratce siatki jest zapisana wartość wyjść). Ponieważ funkcje te są kombinacyjne, to w ich opisie nie uwzględnia się górnego indeksu dotyczącego przedziału czasu.

Korekcja błędnych stanów układu powinna zapewnić ustawienie poprawnego stanu układu po wykryciu błędnego stanu. Można wyróżnić dwie wersje korekcji:

a) synchroniczną - wtedy błąd trwa aż do kolejnego impulsu zegarowego (aktywnego zbocza zegara),

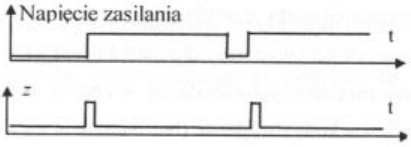
b) asynchroniczną - wtedy czas trwania błędu będzie zależał od opóźnienia elementów układu.

W wersji rozwiązania bez korekcji błędnych stanów i w wersji z synchroniczną ich korekcją wejścia ustawiające przerzutników są nieaktywne.

Ad a) Dla uwzględnienia synchronicznej korekcji błędnych stanów należy w rozwiązaniu nie zawierającym tej korekcji zmodyfikować funkcje wzbudzeń przerzutników i ewentualnie funkcje wyjść. W tym celu należy uzupełnić siatkę przejść/wyjść o przejścia wynikające z synchronicznej korekcji błędnych stanów. Zgodnie z nią określić nowe wyrażenia dla wejść informacyjnych przerzutników i wyjść.

Ad b) Dla realizacji asynchronicznej korekcji błędnych stanów należy określić rozwiązanie dla wejść ustawiających: $\bar{s}, \bar{r}$. Rozwiązanie to powinno zapewnić asynchroniczne wymuszenie przejścia z błędnego stanu do poprawnego. Rozwiązanie dla wejść informacyjnych przerzutników pozostaje takie jak dla przypadku bez korekcji błędnych stanów.

Ustawianie stanu początkowego układu ma na celu wymuszenie stanu poprawnego początkowego po pojawieniu się napięcia zasilania. Trzeba więc dysponować dodatkowym sygnałem z sygnalizującym pojawienie się napięcia zasilania. Sygnał ten może pochodzić z układu wykrywającego pojawienie się napięcia zasilania (rys. 10-13) lub po prostu z przycisku, którego trzeba użyć aby wygenerować sygnał z.



Rys. 10-13

W oparciu o stan wejść i wartość sygnału z należy wypracować sygnały sterujące wejściami ustawiającymi $\bar{s}, \bar{r}$ zgodnie z tabelą podaną na rys. 10-14.

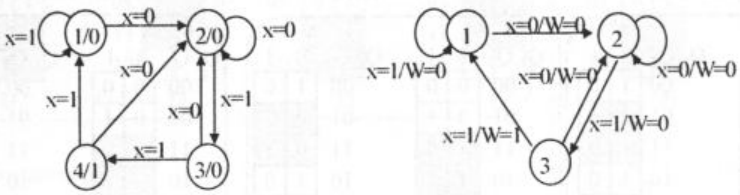
z \ stany wejść	X_1	...	X_n
0	stan nieaktywny $\bar{s}\ \bar{r}$	...	stan nieaktywny $\bar{s}\ \bar{r}$
1	stan aktywny $\bar{s}\ \bar{r}$ wymuszający stan początkowy dla X_1	...	stan aktywny $\bar{s}\ \bar{r}$ wymuszający stan początkowy dla X_n

Rys. 10-14

Przykład 10.1

Zaprojektować układ, który na wyjściu W generuje sygnał W = 1 (na czas nie dłuższy niż okres zegara) wtedy i tylko wtedy, kiedy na wejściu szeregowym x w takt kolejnych impulsów taktujących C pojawia się sekwencja: x = 0→1→1.

Graf przejść: a) *Struktura Moore'a* b) *Struktura Mealy'ego*



Rys. 10-15

W rozwiązaniu układu o strukturze Moore'a (rys. 10-15a) stan wyjścia W = 1 będzie wygenerowany na okres zegara. W rozwiązaniu układu o strukturze Mely'ego (rys. 10-15b) stan wyjścia W = 1 będzie wygenerowany na czas, kiedy układ będąc w stanie numer 3 posiada stan wejścia: x = 1. Zatem, jeżeli zmiana sygnału x następuje między aktywnymi kolejnymi zboczami zegara (np. przy zboczu przeciwnym do aktywnego zbocza projektowanego układu), to stan wyjścia W = 1 będzie wygenerowany na czas krótszy od okresu zegara.

Struktura Moore'a

S^n	x=0	x=1	W
1	2	1	0
2	2	3	0
3	2	4	0
4	2	1	1

S^{n+1}

Struktura Mealy'ego

S^n	x=0	x=1
1	2/0	1/0
2	2/0	3/0
3	2/0	1/1

S^{n+1}/W

Rys. 10-16

Struktura Moore'a

$Q_1^n Q_2^n$	x=0	x=1	W
00	01	00	0
01	01	11	0
11	01	10	0
10	01	00	1

$Q_1^{n+1} Q_2^{n+1}$

Struktura Mealy'ego

$Q_1^n Q_2^n$	x=0	x=1
00	01/0	00/0
01	01/0	11/0
11	01/0	00/1
10		

$Q_1^{n+1} Q_2^{n+1}/W$

Rys. 10-17

Rys. 10-16 przedstawia siatki przejść/wyjść z opisem dziesiętnym stanów układu, a rys. 10-17 binarne (dwójkowe) siatki przejść/wyjść dla struktur: Moore'a i Mealy'ego. Siatkę przejść/wyjść dla układu Mealy'ego można określić w oparciu o graf przejść dla tej struktury albo w wyniku redukcji stanów 1 i 4 w siatce przejść/wyjść dla układu Moore'a.

a)

$Q_1^n Q_2^n$	x^n=0	x^n=1
00	01	00
01	01	11
11	01	10
10	01	00

$Q_1^{n+1} Q_2^{n+1}$

b)

$Q_1^n Q_2^n$	x^n=0	x^n=1
00	01	00
01	01	11
11	01	00
10		

$Q_1^{n+1} Q_2^{n+1}$

c)

$Q_1 Q_2$	W
0 0	0
0 1	0
1 1	0
1 0	1

$W = Q_1 \overline{Q_2}$

d)

$Q_1 Q_2$	x=0	x=1
00	0	0
01	0	0
11	0	1
10		

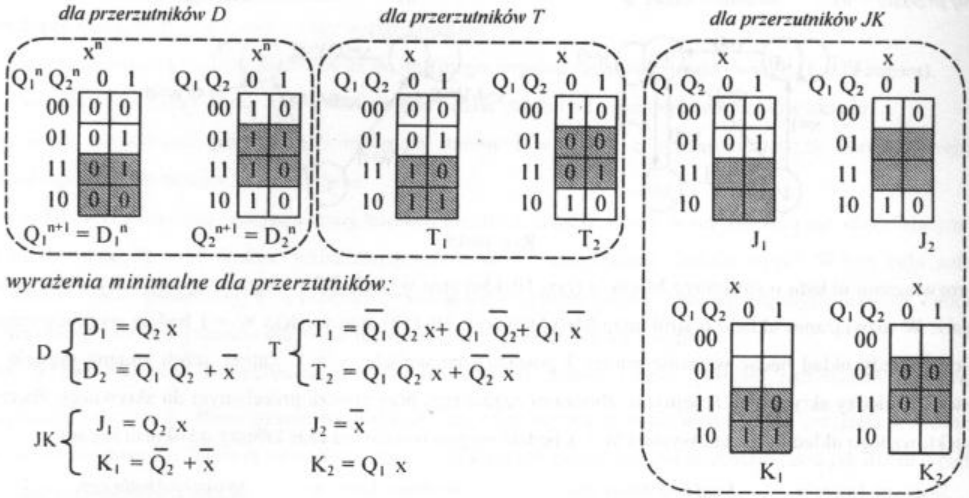
W

$W = Q_1 X$

Rys. 10-18

Blok Pamięci opisany jest siatkami stanów wewnętrznych Q_1 i Q_2 (rys. 10-18a i 10-18b). Blok Kombinacyjny Wyjściowy opisany jest odpowiednio w postaci tablicy zależności dla struktury Moore'a (rys. 10-18 c) i siatki zależności dla struktury Mealy'ego (rys. 10-18d).

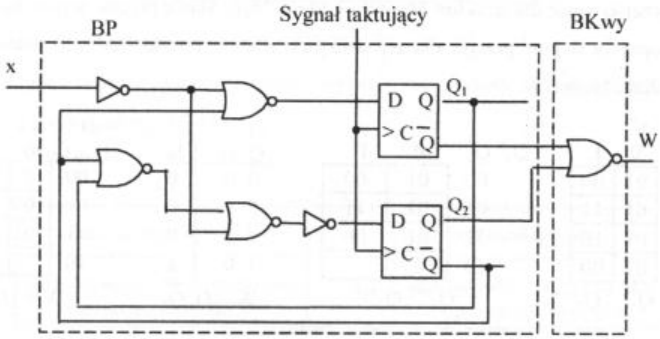
Ciąg dalszy syntezy układu Moore'a (rozwiązanie dla BP) w przypadku zastosowania różnych typów przerzutników przedstawia rys. 10-19.



Rys. 10-19

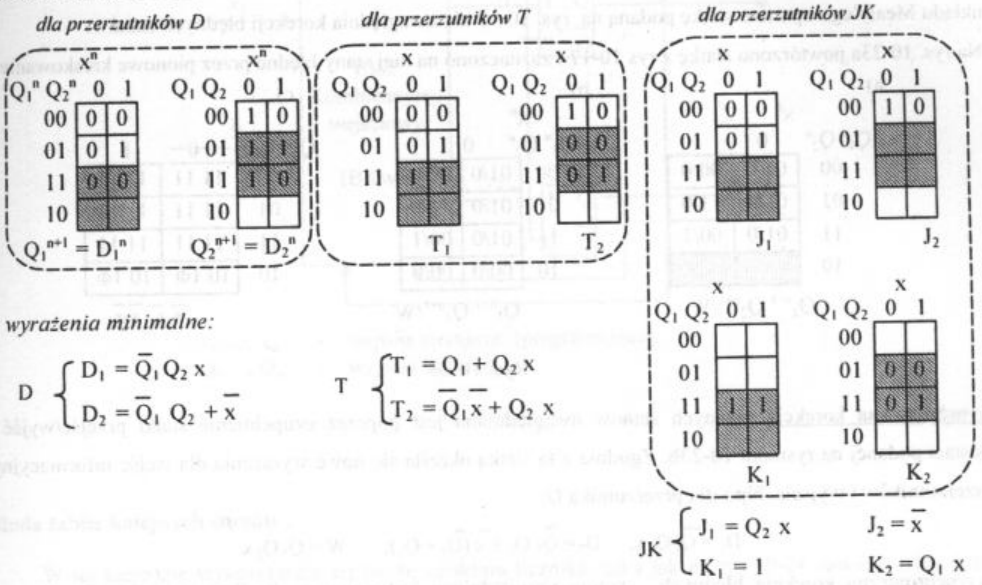
Siatki dla wejść informacyjnych przerzutników można określić bezpośrednio w oparciu o binarne siatki przejść poszczególnych przerzutników. W określaniu ich pomocne jest wyróżnienie stanów, w których $Q^n = 1$ (np. na rys. 10-19 przez zakreskowanie) od stanów w których $Q^n = 0$ (bez zakreskowania). W siatkach i wyrażeniach dla funkcji wejść informacyjnych przerzutników pomija się górny indeks n, gdyż dotyczą one zależności kombinacyjnych.

Schemat realizacji układu o strukturze Moore'a w oparciu o przerzutniki D i bramki NOR podaje rys. 10-20.



Rys. 10-20

Określenie rozwiązania Bloku Pamięci dla struktury Meay'ego w oparciu o przerzutniki typu D i T przedstawia rys. 10-21.



wyrażenia minimalne:

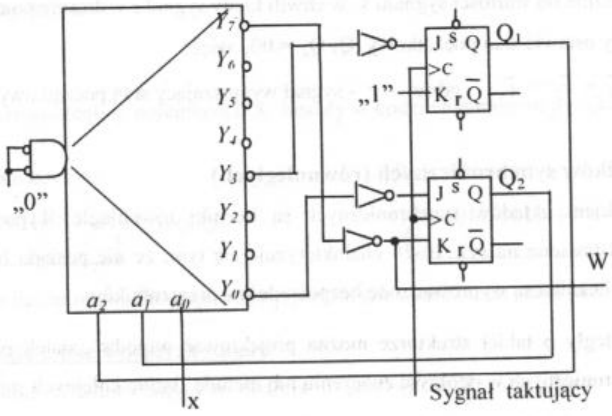
$$D \begin{cases} D_1 = \overline{Q}_1 Q_2 x \\ D_2 = \overline{Q}_1 Q_2 + \overline{x} \end{cases} \qquad T \begin{cases} T_1 = Q_1 + Q_2 x \\ T_2 = \overline{Q}_1 \overline{x} + Q_2 x \end{cases}$$

$$JK \begin{cases} J_1 = Q_2 x & J_2 = \overline{x} \\ K_1 = 1 & K_2 = Q_1 x \end{cases}$$

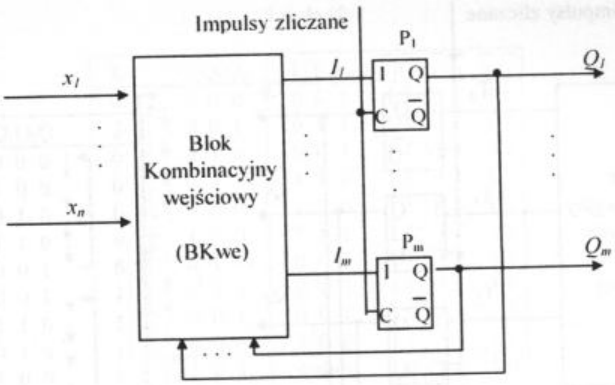
Rys. 10-21

Dla określenia rozwiązania BKwe w oparciu o demultiplekser (rys. 10-22) potrzebne są wyrażenia w postaci kanonicznej:

$$\begin{cases} J_1 = \Sigma(3)_{Q_1 Q_2 x} \\ K_1 = \Sigma(6,7)_{Q_1 Q_2 x} \text{ lub } K_1 = 1 \end{cases} \qquad \begin{cases} J_2 = \Sigma(0)_{Q_1 Q_2 x} \\ K_2 = \Sigma(7)_{Q_1 Q_2 x} \end{cases} \qquad W = \Sigma(7)_{Q_1 Q_2 x}$$



Rys. 10-22



$x_1, \ldots, x_n$ - wejścia sterujące (programujące).
$Q_1, \ldots, Q_m$ - wyjścia licznika.

Rys. 10-24

Metoda tablic kolejnych stanów

W tej metodzie wykorzystuje się to, że struktura licznika taka jak na rys 10-24 zawiera tylko Blok Pamięci. Synteza sprowadza się więc do zaprojektowania Bloku Kombinacyjnego wejściowego sterującego wejściami informacyjnymi przerzutników wybranych do realizacji licznika. BKwe można zaprojektować w oparciu o taką tablicę zależności, w której kombinacje argumentów funkcji: $x_1 .. x_n Q_1 .. Q_m$ w lewej części tablicy nie są zapisane w dowolnej kolejności, tylko w kolejności odpowiadającej kolejnym stanom licznika dla różnych wartości wejść programujących. Taką tablicę będziemy nazywać tablicą kolejnych stanów.

Wartości wejść informacyjnych w tej tablicy można łatwo określić na podstawie porównania stanów licznika: aktualnego i następnego.

Przykład 10.2

Zaprojektować licznik równoległy o pojemności: 5, liczący w kodzie naturalnym dwójkowym:

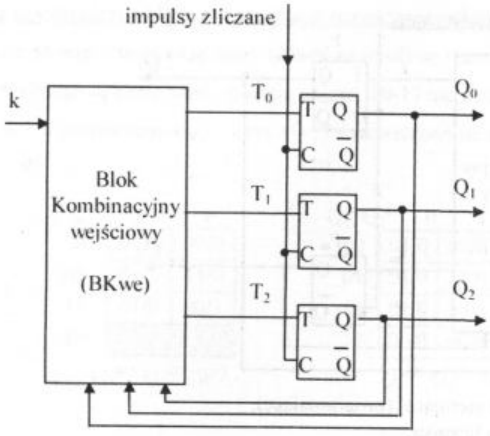
dla k = 0 w przód, czyli: i dla k = 1: w tył, czyli:

$(000 \rightarrow 001 \rightarrow 010 \rightarrow 011 \rightarrow 100)_{Q_2 Q_1 Q_0}$ $(100 \rightarrow 011 \rightarrow 010 \rightarrow 001 \rightarrow 000)_{Q_2 Q_1 Q_0}$

(pojemność licznika to liczba impulsów zegarowych, po których wraca on do stanu początkowego)

Synteza licznika metodą tablic kolejnych stanów

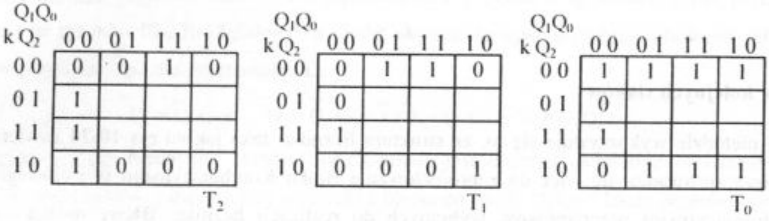
Ogólny schemat licznika zrealizowanego w oparciu o przerzutnik T przedstawia rys. 10-25. W tablicy kolejnych stanów (tabela 10-1) wartości wejść informacyjnych T można łatwo określić na podstawie porównania stanów licznika: aktualnego i następnego.



Rys. 10-25

Tabela 10-1

k	$Q_2Q_1Q_0$	$T_2T_1T_0$
0	0 0 0	0 0 1
0	0 0 1	0 1 1
0	0 1 0	0 0 1
0	0 1 1	1 1 1
0	1 0 0	1 0 0
1	1 0 0	1 1 1
1	0 1 1	0 0 1
1	0 1 0	0 1 1
1	0 0 1	0 0 1
1	0 0 0	1 0 0



Rys. 10-26

Z tablicy kolejnych stanów (tabela 10-1) wynikają siatki (rys. 10-26) oraz wyrażenia:

$$T_2 = \bar{k}\,Q_1\,Q_0 + k\,\bar{Q}_1\,\bar{Q}_0 + Q_2 \qquad T_1 = \bar{k}\,Q_0 + k\,Q_2 + k\,Q_1\,\bar{Q}_0$$

$$T_0 = \bar{k}\,\bar{Q}_2 + k\,Q_2 + Q_1 + Q_0$$

Podane rozwiązania nie uwzględnia korekcji błędnych stanów.

Asynchroniczna korekcja błędnych stanów wymaga uzupełnienia poprzedniego rozwiązania o układ sterujący wejściami ustawiającymi przerzutników. Wyrażenia dla wejść informacyjnych przerzutników pozostają bez zmian.

Tabela 10-2 oprócz przejść wynikających z programu pracy układu uwzględnia też przejścia wynikające z korekcji błędnych stanów. Wynikają z niej wyrażenia dla wejść ustawiających przerzutników:

$$\bar{s}_2 = \bar{s}_1 = \bar{s}_0 = 1$$

$$\bar{r}_2 = \overline{\bar{k}\,Q_2\,(Q_1 + Q_0)} = k + \overline{Q_2\,(Q_1 + Q_0)} = k + \bar{r}_1$$

$$\bar{r}_1 = \bar{r}_0 = \overline{Q_2\,(Q_1 + Q_0)}$$

Tabela 10-2

k	$Q_2Q_1Q_0$	$T_2T_1T_0$	$\overline{S_2}\overline{r_2}$	$\overline{S_1}\overline{r_1}$	$\overline{S_0}\overline{r_0}$
0	0 0 0	0 0 1	1 1	1 1	1 1
0	0 0 1	0 1 1	1 1	1 1	1 1
0	0 1 0	0 0 1	1 1	1 1	1 1
0	0 1 1	1 1 1	1 1	1 1	1 1
0	1 0 0	1 0 0	1 1	1 1	1 1
1	1 0 0	1 1 1	1 1	1 1	1 1
1	0 1 1	0 0 1	1 1	1 1	1 1
1	0 1 0	0 1 1	1 1	1 1	1 1
1	0 0 1	0 0 1	1 1	1 1	1 1
1	0 0 0	1 0 0	1 1	1 1	1 1
0	1 0 1	ΦΦΦ	1 0	1Φ	1 0
0	1 1 0	ΦΦΦ	1 0	1 0	1Φ
0	1 1 1	ΦΦΦ	1 0	1 0	1 0
1	1 0 1	ΦΦΦ	Φ1	1Φ	1 0
1	1 1 0	ΦΦΦ	Φ1	1 0	1Φ
1	1 1 1	ΦΦΦ	Φ1	1 0	1 0

Synchroniczna korekcja błędnych stanów wymaga modyfikacji poprzedniego rozwiązania pod względem realizacji wejść informacyjnych przerzutników.

Tabela 10-3 zawiera tablicę kolejnych stanów uwzględniającą przejścia wynikające z korekcji błędnych stanów.

Tabela 10-3

k	$Q_2Q_1Q_0$	$T_2T_1T_0$
0	0 0 0	0 0 1
0	0 0 1	0 1 1
0	0 1 0	0 0 1
0	0 1 1	1 1 1
0	1 0 0	1 0 0
1	1 0 0	1 1 1
1	0 1 1	0 0 1
1	0 1 0	0 1 1
1	0 0 1	0 0 1
1	0 0 0	1 0 0
0	1 0 1	1 0 1
0	1 1 0	1 1 0
0	1 1 1	1 1 1
1	1 0 1	0 0 1
1	1 1 0	0 1 0
1	1 1 1	0 1 1

Q_1Q_0 / k Q_2

k Q_2 \ Q_1Q_0	00	01	11	10
00	0	0	1	0
01	1	1	1	1
11	1	0	0	0
10	1	0	0	0

T_2

Q_1Q_0 / k Q_2

k Q_2 \ Q_1Q_0	00	01	11	10
00	0	1	1	0
01	0	0	1	1
11	1	0	1	1
10	0	0	0	1

T_1

Q_1Q_0 / k Q_2

k Q_2 \ Q_1Q_0	00	01	11	10
00	1	1	1	1
01	0	1	1	0
11	1	1	1	0
10	0	1	1	1

T_0

Rys. 10-27

Z siatek na rys. 10-27 wynikają zmodyfikowane wyrażenia dla wejść informacyjnych przerzutników:

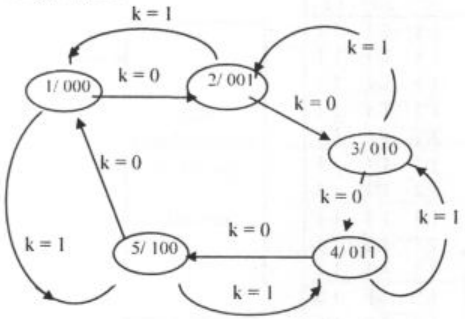
$$T_2 = k\,Q_1\,\overline{Q_0} + k\,Q_1\,Q_0 + k\,\overline{Q_2}$$

$$T_1 = \overline{k}\,Q_1\,\overline{Q_0} + k\,Q_2\,\overline{Q_0} + \overline{k}\,\overline{Q_2}\,Q_0 + Q_2\,Q_1$$

$$T_0 = \overline{k}\,\overline{Q_2} + k\,\overline{Q_2}\,Q_1 + \overline{Q_2}\,Q_1 + Q_0$$

Synteza tego samego licznika metodą siatek przejść

Graf przejść:



Rys. 10-28

Siatka przejść/wyjść:

S^n	k^n = 0	k^n = 1	$Q_2Q_1Q_0$
1	2	5	0 0 0
2	3	1	0 0 1
3	4	2	0 1 0
4	5	3	0 1 1
5	1	4	1 0 0

S^{n+1}

Rys. 10-29

Tabela 10-4

S	$Q_2 Q_1 Q_0$
1	0 0 0
2	0 0 1
3	0 1 0
4	0 1 1
5	1 0 0

Przy kodowaniu stanów należy uwzględnić wartości sygnałów wyjściowych i nimi zakodować stany licznika (tabela 10-4). Wtedy struktura będzie taka, jak na rys. 10-25. W opisie pracy układu górny indeks dotyczący przedziału czasu występuje tylko wtedy, kiedy opis dotyczy zmian stanu układu.

Binarna siatka przejść/wyjść

(S^n)	$Q_2^n Q_1^n Q_0^n$	k^n = 0	k^n = 1
(1)	000	0 0 1	1 0 0
(2)	001	0 1 0	0 0 0
(4)	011	1 0 0	0 1 0
(3)	010	0 1 1	0 0 1
	110		
	111		
	101		
(5)	100	0 0 0	0 1 1

$Q_2^{n+1} Q_1^{n+1} Q_0^{n+1} = D_2^n D_1^n D_0^n$

Siatka wejść informacyjnych przerzutników T

$Q_2 Q_1 Q_0$	k = 0	k = 1
000	0 0 1	1 0 0
001	0 1 1	0 0 1
011	1 1 1	0 0 1
010	0 0 1	0 1 1
110		
111		
101		
100	1 0 0	1 1 1

$T_2 T_1 T_0$

Rys. 10-30

Dla wejść przerzutników D otrzymujemy wyrażenia:

$$D_2 = \overline{k}\, Q_1\, Q_0 + k\, \overline{Q_2}\, \overline{Q_1}\, \overline{Q_0} \qquad D_1 = \overline{k}\, \overline{Q_1}\, Q_0 + \overline{k}\, Q_1\, \overline{Q_0} + k\, Q_1\, Q_0 + k\, Q_2$$

$$D_0 = \overline{k}\, \overline{Q_2}\, \overline{Q_0} + Q_1\, \overline{Q_0} + k\, Q_2$$

Dla przerzutników T otrzymujemy wyrażenia (takie jakie uzyskano stosując metodę tablicy kolejnych stanów):

$$T_2 = \overline{k}\, Q_1 Q_0 + \overline{k}\, \overline{Q_1}\, Q_0 + Q_2 \qquad T_1 = \overline{k}\, Q_0 + k Q_2 + k\, Q_1\, \overline{Q_0} \qquad T_0 = \overline{k}\, \overline{Q_2} + k\, Q_2 + Q_1 + Q_0$$

Siatki dla wejść informacyjnych przerzutników JK

$Q_2Q_1Q_0$	k=0	k=1
000	0 Φ	1 Φ
001	0 Φ	0 Φ
011	1 Φ	0 Φ
010	0 Φ	0 Φ
110		
111		
101		
100	Φ 1	Φ 1

J_2K_2

$Q_2Q_1Q_0$	k=0	k=1
000	0 Φ	0 Φ
001	1 Φ	0 Φ
011	Φ 1	Φ 0
010	Φ 0	Φ 1
110		
111		
101		
100	0 Φ	1 Φ

J_1K_1

$Q_2Q_1Q_0$	k=0	k=1
000	1 Φ	0 Φ
001	Φ 1	Φ 1
011	Φ 1	Φ 1
010	1 Φ	1 Φ
110		
111		
101		
100	0 Φ	1 Φ

J_0K_0

Rys. 10-31

Dla przerzutników JK otrzymujemy rozwiązanie:

$$J_2 = \overline{k}\,Q_1Q_0 + \overline{k}\,\overline{Q_1}\,\overline{Q_0} \qquad J_1 = \overline{k}\,Q_0 + kQ_2 \qquad J_0 = \overline{k}\,\overline{Q_2} + kQ_2 + Q_1$$

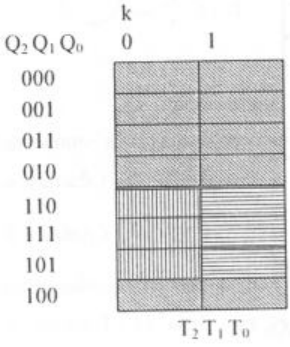
$$K_2 = 1 \qquad K_1 = \overline{k}\,Q_0 + \overline{k}\,Q_0 \qquad K_0 = 1$$

Dotychczasowe rozwiązania uzyskane metoda siatek przejść nie uwzględniają korekcji błędnych stanów.

Rozwiązanie korekcji błędnych stanów dla licznika projektowanego metodą siatek przejść

Po wykryciu błędnego stanu licznika nie należącego do kodu zliczania dla danego stanu wejścia programującego k, należy ustawić stan początkowy dla tego stanu k.

W obydwu wersjach korekcji (asynchronicznej i synchronicznej) wymagania można przedstawić w postaci siatki podanej na rys. 10-32.



Objaśnienia:

– błędny stan, który trzeba skorygować na stan: $Q_2Q_1Q_0 = 000$

– błędny stan, który trzeba skorygować na stan $Q_2Q_1Q_0 = 100$

– poprawny stan, którego nie trzeba korygować.

Rys. 10-32

Asynchroniczna korekcja błędnych stanów wymaga jedynie uzupełnienia poprzedniego rozwiązania o układ odpowiednio sterujący wejściami ustawiającymi przerzutników. Układ ten jest niezależny od typu zastosowanych przerzutników synchronicznych: D, T, czy JK. Opisuje go siatka dla wejść ustawiających podana na rys. 10-33.

$Q_2Q_1Q_0$	k = 0	k = 1
0 0 0	11 11 11	11 11 11
0 0 1	11 11 11	11 11 11
0 1 1	11 11 11	11 11 11
0 1 0	11 11 11	11 11 11
1 1 0	10 10 1Φ	Φ1 10 1Φ
1 1 1	10 10 10	Φ1 10 10
1 0 1	10 1Φ 10	Φ1 1Φ 10
1 0 0	11 11 11	11 11 11
	$\bar{s}_2\bar{r}_2$ $\bar{s}_1\bar{r}_1$ $\bar{s}_0\bar{r}_0$	

Z siatki wynikają wyrażenia:

$$\bar{s}_2 = \bar{s}_1 = \bar{s}_0 = 1$$

$$\bar{r}_2 = \overline{k\,Q_2\,(Q_1 + Q_0)} = k + \overline{Q_2\,(Q_1 + Q_0)} = k + \bar{r}_1$$

$$\bar{r}_1 = \bar{r}_0 = \overline{Q_2\,(Q_1 + Q_0)}$$

Rys. 10-33

Synchroniczna korekcja błędnych stanów wymaga modyfikacji rozwiązania dla wejść informacyjnych przerzutników tak, jak to podaje binarna siatka przejść/wyjść podana na rys. 10-34.

$Q_2^n Q_1^n Q_0^n$	k = 0	k = 1
0 0 0	0 0 1	1 0 0
0 0 1	0 1 0	0 0 0
0 1 1	1 0 0	0 1 0
0 1 0	0 1 1	0 0 1
1 1 0	0 0 0	1 0 0
1 1 1	0 0 0	1 0 0
1 0 1	0 0 0	1 0 0
1 0 0	0 0 0	0 1 1
	$Q_2^{n+1} Q_1^{n+1} Q_0^{n+1} = D_2^n D_1^n D_0^n$	

Rys. 10-34

Funkcje wejść informacyjnych dla przerzutnika D przyjmują wtedy postać:

$$D_2 = \bar{k}\,\bar{Q}_2\,Q_1\,Q_0 + k\,\bar{Q}_2\,\bar{Q}_1\,\bar{Q}_0 + kQ_2\,Q_1 + kQ_2\,Q_0$$

$$D_1 = \bar{k}\,\bar{Q}_2\,\bar{Q}_1\,Q_0 + \bar{k}\,\bar{Q}_2\,Q_1\,\bar{Q}_0 + k\,\bar{Q}_2\,Q_1\,Q_0 + k\,Q_2\,\bar{Q}_1\,\bar{Q}_0$$

$$D_0 = \bar{k}\,\bar{Q}_2\,\bar{Q}_0 + \bar{Q}_2\,Q_1\,\bar{Q}_0 + k\,Q_2\,\bar{Q}_1\,\bar{Q}_0$$

W podobny sposób należy zmienić rozwiązanie oparte o przerzutniki T lub JK.

10.4. Przykłady syntezy wybranych układów arytmetycznych

Przykład 10.3

Zaprojektować sumator szeregowy dla liczb dwójkowych. Sumator szeregowy jest typowym układem synchronicznym.

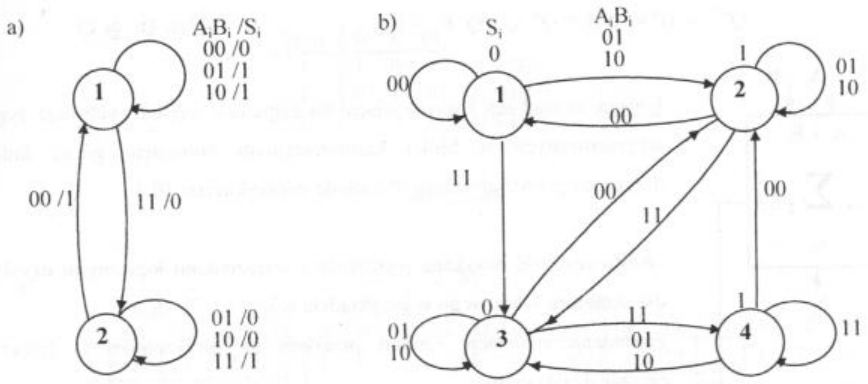
W sumatorze szeregowym dodawane liczby są wprowadzane do sumatora bit po bicie, poczynając od bitu najmniej znaczącego. W układzie takim występuje konieczność odróżniania jednakowych stanów wyjść. Na rys. 10-35 przedstawiono grafy przejść sumatora szeregowego:

a) dla struktury Mealy'ego,

b) dla struktury Moore'a.

Dla pełniejszego zrozumienia odmienności przedstawionych grafów można zaproponować następującą ich interpretację.

W przypadku układu o strukturze Mealy'ego stan 1 odpowiada przypadkowi, gdy $C_i = 0$, natomiast stan 2 odpowiada przypadkowi, gdy $C_i = 1$.



Rys. 10-35

Dla układu Moore'a przyporządkowanie to jest następujące:

w stanach 1 i 2 : $C_i = 0$,

w stanach 3 i 4 : $C_i = 1$.

Stan wyjść układu w tych stanach jest następujący:

w stanach 1 i 3 : $S_i = 0$

w stanach 2 i 4 : $S_i = 1$

Kolejne etapy syntezy sumatora szeregowego jako układu Mealy'ego pokazano na rys. 10-36.

a)

S^n \ $A_i B_i$	00	01	11	10
1	1/0	1/1	2/0	1/1
2	1/1	2/0	2/1	2/0

S^{n+1} / S_i

b)

Q^n \ $A_i B_i$	00	01	11	10
1	0/0	0/1	1/0	0/1
2	1/0	1/0	1/1	1/0

Q^{n+1}/S_i

c)

Q^n \ $A_i B_i$	00	01	11	10
0	0	0	1	0
1	0	1	1	1

Q^{n+1}

d)

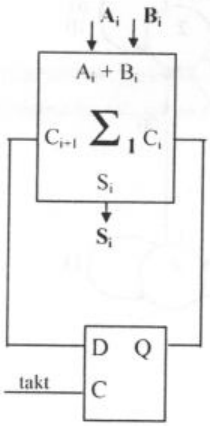
Q \ $A_i B_i$	00	01	11	10
0	0	1	0	1
1	1	0	1	0

S_i

Rys. 10-36

Z tablic uzyskuje się odpowiednio funkcję przejść i funkcję wyjścia sumatora:

$$Q^{n+1} = D^n = A_i B_i + Q^n A_i + Q^n B_i = C_{i+1} \qquad S_i = A_i \oplus B_i \oplus Q$$



Rys. 10-37

Uwaga. w siatkach i wyrażeniach dla sygnałów wejściowych oraz sygnałów wypracowanych w bloku kombinacyjnym pominięto górny indeks n dotyczący przedziału czasu. Podobnie w przykładzie 10.4.

Warto zestawić uzyskane wyrażenia z wyrażeniami logicznymi uzyskanymi dla sumatora 1-bitowego w przykładzie 5.3.

Zakładając realizację funkcji przejścia na przerzutniku D (przerzutnik przeniesienia) mamy:

$D^n = Q^{n+1} = A_i B_i + Q^n A_i + Q^n B_i$

Z postaci uzyskanych funkcji wynika, ze łatwo zrealizować układ z zastosowaniem sumatora 1-bitowego. Schemat logiczny sumatora szeregowego przedstawiono na rys. 10-37.

Przykład 10.4.

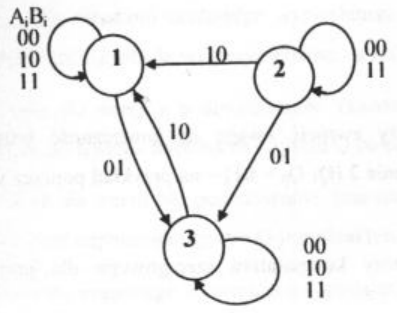
Zaprojektować komparator szeregowy dla liczb dwójkowych.

W przypadku szeregowego przetwarzania informacji do porównywania liczb stosuje się komparator szeregowy. Porównanie liczb może odbywać się począwszy od bitów najmniej znaczących (rys. 10-38a - graf przejść układu Moore'a) lub bitow najbardziej znaczących (rys. 10-38b – graf przejść układu Moore'a). W pierwszym przypadku liczbą większą jest liczba mająca większą wartość ostatniego bitu. Natomiast

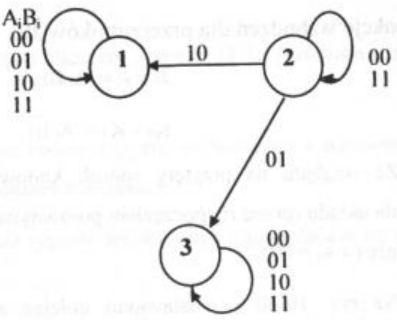
w drugim przypadku proces porównywania może zostać przerwany w chwili wystąpienia różnicy między bitami – większa jest liczba mająca większy pierwszy bit.

Kolejne etapy syntezy komparatora szeregowego z porównywaniem bitów od najmniej znaczących pokazano na rys.10-39.

a)



b)



c)

Stan	W	R	M	
1	1	0	0	(A > B)
2	0	1	0	(A = B)
3	0	0	1	(A < B)

Rys. 10-38

a)

S^n	00	01	11	10	W	R	M
1	1	3	1	1	1	0	0
2	2	3	2	1	0	1	0
3	3	3	3	1	0	0	1

A_iB_i ; S^{n+1}

b)

$Q_1^nQ_0^n$	00	01	11	10	W	R	M
(1) 00	00	11	00	00	1	0	0
(2) 01	01	11	01	00	0	1	0
(3) 11	11	11	11	00	0	0	1
10							

A_iB_i ; Q_1^{n+1} Q_0^{n+1}

c)

$Q_1^nQ_0^n$	00	01	11	10
00	0	1	0	0
01	0	1	0	0
11	1	1	1	0
10				

A_iB_i ; Q_1^{n+1}

d)

$Q_1^nQ_0^n$	00	01	11	10
00	0	1	0	0
01	1	1	1	0
11	1	1	1	0
10				

A_iB_i ; Q_0^{n+1}

Rys. 10-39

Z siatek wynika następująca postać funkcji przełączających:

- funkcje wyjścia:

$$W = \overline{Q_0}, \quad R = \overline{Q_1}\,Q_0, \quad M = Q_0,$$

- funkcje wzbudzeń dla przerzutników JK:

$$J_0 = J_1 = \overline{A_i}\,B_i,$$

$$K_0 = K_1 = A_i\,\overline{B_i}.$$

Ze względu na przyjęty sposób kodowania należy zwrócić uwagę na konieczność wstępnego ustawienia układu (przed rozpoczęciem porównywania) w stanie 2 ($Q_1\,Q_0 = 01$) – na przykład poprzez wejście ustawiające ($r_1 s_0 = 00$).

Na rys. 10-40 przedstawiono kolejne etapy syntezy komparatora szeregowego dla przypadku porównywania bitów począwszy od najbardziej znaczących.

a)

S^n \ A_iB_i	00	01	11	10	W	R	M
1	1	1	1	1	1	0	0
2	2	3	2	1	0	1	0
3	3	3	3	3	0	0	1

S^{n+1}

b)

$Q_1{}^nQ_0{}^n$ \ A_iB_i	00	01	11	10	W	R	M
(1) 00	00	00	00	00	1	0	0
(2) 01	01	11	01	00	0	1	0
(3) 11	11	11	11	11	0	0	1
10							

$Q_1{}^{n+1}$ $Q_0{}^{n+1}$

c)

$Q_1{}^nQ_0{}^n$ \ A_iB_i	00	01	11	10
00	0	0	0	0
01	0	1	0	0
11	1	1	1	1
10				

$Q_1{}^{n+1}$

d)

$Q_1{}^nQ_0{}^n$ \ A_iB_i	00	01	11	10
00	0	0	0	0
01	1	1	1	0
11	1	1	1	1
10				

$Q_0{}^{n+1}$

Rys. 10-40

W tym przypadku uzyskuje się następujące funkcje przełączające:

- funkcje wyjścia: $W = \overline{Q_0}, \quad R = \overline{Q_1}\,Q_0, \quad M = Q_1,$
- funkcje wzbudzeń dla przerzutników JK:

$$J_1 = Q_0\,\overline{A_i}\,B_i, \qquad K_1 = 0,$$

$$J_0 = 0, \qquad K_0 = \overline{Q_1}\,A_i\,\overline{B_i}.$$

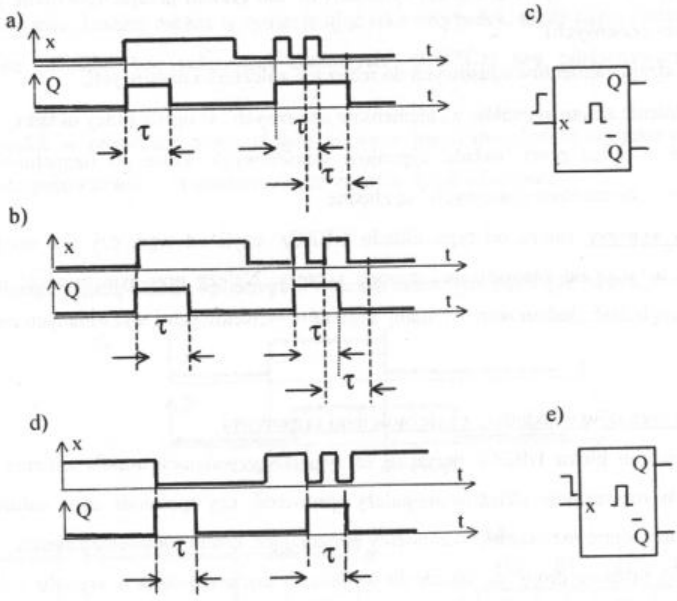
11. UKŁADY Z ZALEŻNOŚCIAMI CZASOWYMI

W programie pracy układów cyfrowych mogą występować zależności czasowe. Do ich realizacji w układach cyfrowych stosuje się tzw. „elementy czasowe" generujące pojedynczy impuls wyzwalany narastającym lub opadającym zboczem sygnału wejściowego.

Dla elementu czasowego wyzwalanego narastającym zboczem rysunek 11-1c przedstawia symbol, a jego działanie ilustrują wykresy czasowe na rysunkach:

- 11-1a dla wersji z podtrzymaniem (każde narastające zbocze sygnału wejściowego x pojawiające się podczas trwania impulsu na wyjściu Q powoduje przedłużenie impulsu),
- 11-1b dla wersji bez podtrzymania (narastające zbocze sygnału wejściowego x pojawiające się podczas trwania impulsu na wyjściu Q jest nieaktywne).

Dla elementu czasowego wyzwalanego opadającym zboczem sygnału wejściowego x rysunek 11-1e przedstawia symbol, a jego działanie w wersji bez podtrzymania ilustrują wykresy czasowe na rysunku 11-1d.

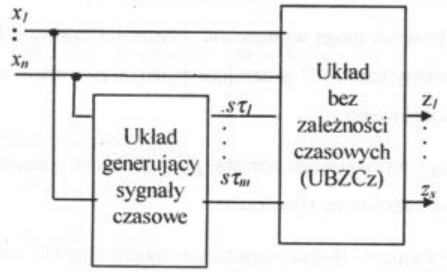


Rys. 11-1

W oparciu o „elementy czasowe" i zestaw funktorów tworzący system funkcjonalnie pełny można realizować układ z zależnościami czasowymi[1].

[1] Dokładniej opisane w ćw. 8 [6].

Struktura układu z zależnościami czasowymi przedstawiona jest na rysunku 11-2.



Rys. 11-2

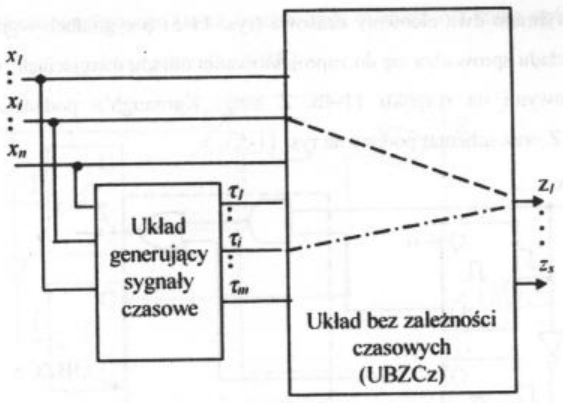
Algorytm syntezy układu z zależnościami czasowymi w porównaniu z algorytmem syntezy układów bez zależności czasowych jest uzupełniony o etapy początkowe.

- **etapy początkowe**:
  - opis pracy układu: wykresem czasowym, słowny lub grafem przejść (podobnie jak dla układu bez zależności czasowych),
  - wybór rodzaju elementów czasowych do realizacji zależności czasowych,
  - uwzględnienie zmian sygnałów z „elementów czasowych" w opisie pracy układu,
  - usunięcie z opisu pracy układu sygnałów wejściowych, które po uzupełnieniu opisu zbiorem sygnałów z „elementów czasowych" są zbędne.
- **Dalsze etapy syntezy** zależą od typu układu UBZCz, czyli od tego, czy jest on kombinacyjny czy sekwencyjny, a także od zastosowanej metody syntezy. Należy przy tym zwrócić uwagę na to, żeby poprawnie uwzględnić „jednoczesne" zmiany sygnałów wejściowych i wyzwalanych nimi „sygnałów czasowych".

Jednoczesna zmiana sygnałów w układzie z zależnościami czasowymi

Jeżeli przy projektowaniu bloku UBZCz przyjęto, że sygnał wyzwalający impuls zmienia się wcześniej niż sygnał czasowy, to w rozwiązaniu układowym należy sprawdzić, czy spełnione są te założenia. Jeżeli nie są spełnione, to należy uzupełnić rozwiązanie o elementy opóźniające, celem spełnienia założeń.

Rysunek 11-3 ilustruje drogi, po jakich do wyjścia Z_1 docierają zmiany sygnału x_i (linia kreskowana) i zmiany sygnału czasowego τ_i wyzwolonego przez: x_i (linia kreska-kropka). Chociaż zmiana sygnału x_i wyzwalającego impuls czasowy „dociera" do wejścia bloku UBZCz z mniejszym opóźnieniem niż zmiana impulsu τ_i wygenerowanego przez x_i, to jednak nie można ogólnie wykluczyć, że opóźnienie sygnału x_i na drodze zaznaczonej linią kreskowaną może być większe od opóźnienia sygnału τ_i na drodze zaznaczonej linią kreska-kropka.



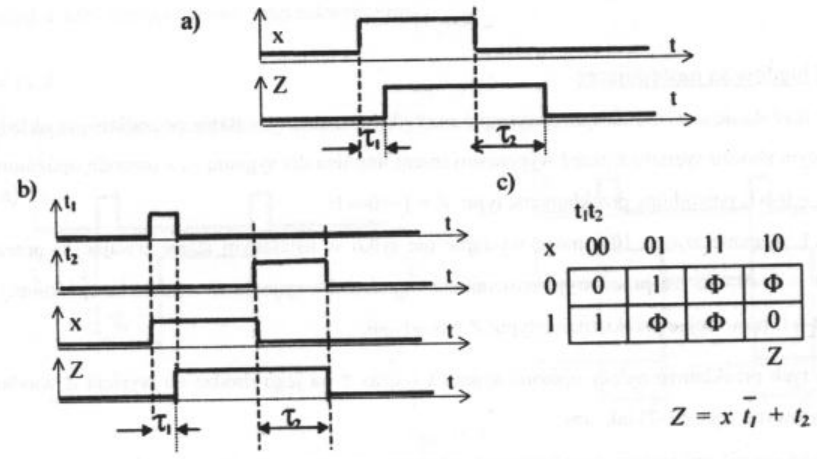
Rys. 11-3

Dla zapewnienia poprawnej pracy układu niezależnie od opóźnień elementów, czyli takiej która nie będzie wymagała sprawdzenia opóźnień i ewentualnej korekty, w programie pracy układu rozważane zmiany sygnałów: x_i i τ_i należy uwzględnić, tak jak uwzględnia się jednoczesną zmianę wejść. Jest to szczególnie ważne w układzie sekwencyjnym. Inaczej można tę sytuację ująć jako przypadek, kiedy okres zmian sygnałów: x_i τ_i jako wejściowych do „Układu bez zależności czasowych" (UBZCz) jest porównywalny z opóźnieniami tego układu.

Taki przypadek w programie pracy układu sekwencyjnego uwzględnia się jako różne rodzaje pracy układu: szybciej następuje zmiana x_i w porównaniu ze zmianą τ_i lub odwrotnie.

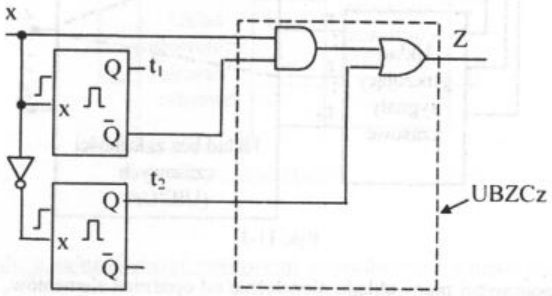
Przykład 11.1

Należy przeprowadzić syntezę układu opisanego wykresem czasowym podanym na rys. 11-4a.



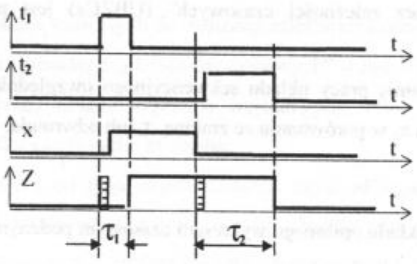
Rys. 11-4

Do realizacji opóźnień wybrano dwa elementy czasowe (rys. 11-5) o sygnałach wyjściowych t_1 i t_2. Po ich uwzględnieniu synteza układu sprowadza się do zaprojektowania układu o wejściach: x, t_1, t_2 i wyjściu Z, opisanego wykresami czasowymi na rysunku 11-4b. Z siatki Karnaugh'a podanej na rys 11-4c wynika minimalne wyrażenie dla Z oraz schemat podany na rys. 11-5.



Rys. 11-5

Rozwiązanie wg schematu podanego na rys. 11-5 obarczone jest możliwością wystąpienia chwilowych błędów zaznaczonych kreskowanym polem na wykresie (rys. 11-6).



Rys. 11-6

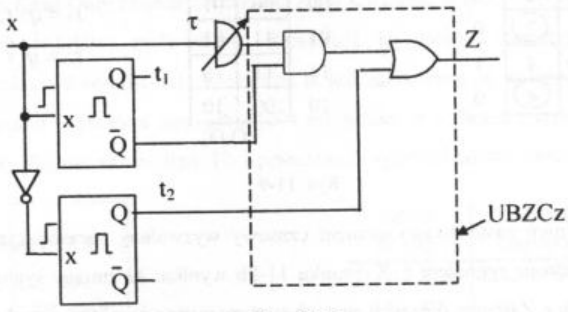
Możliwości błędów są następujące:

- wartość $Z = 0$ w stanie $xt_1t_2 = 000$, może wystąpić nie tylko w ustalonym stanie początkowym układu ale też przy opadającym zboczu sygnału x przed wygenerowaniem impulsu dla sygnału t_2, z powodu opóźnionej zmiany sygnału $t_2 = 0 \rightarrow 1$, powodując przekłamanie typu: $Z = 1 \rightarrow 0 \rightarrow 1$.
- wartość $Z = 1$ w stanie $xt_1t_2 = 100$, może wystąpić nie tylko w ustalonym stanie trwającym przez czas t_2 po zmianie $x = 1 \rightarrow 0$, ale też przed wygenerowaniem impulsu dla sygnału t_1 z powodu opóźnionej zmiany sygnału $t_1 = 0 \rightarrow 1$, powodując przekłamanie typu: $Z = 0 \rightarrow 1 \rightarrow 0$.

Dla uniknięcia tych przekłamań należy opóźnić sygnał x o czas τ na jego drodze do wyjścia Z wiodącej poza elementami czasowymi (rys. 11-7) tak, aby:

- czas τ był większy niż opóźnienie wnoszone przez element czasowy generujący sygnał t_1.

- suma czasu τ i opóźnienia wnoszonego przez iloczyn była większa niż suma opóźnień wnoszonych przez element czasowy generujący sygnał t_2 i bramkę NOT.



Rys. 11-7

Dyskusja wersji rozwiązania układu pracującego bez przekłamań niezależnie od opóźnień elementów

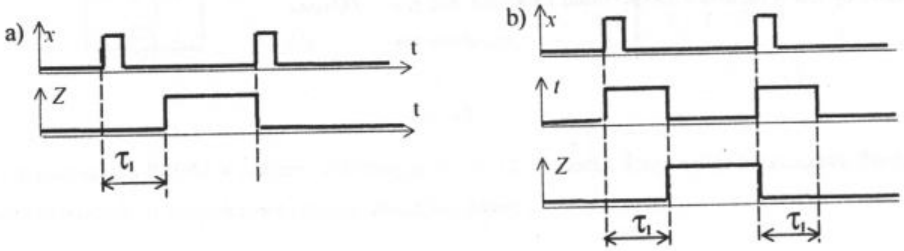
Zgodnie w uwagami dotyczącymi jednoczesnej zmiany sygnału wyzwalającego impuls czasowy i zmiany impulsu czasowego, chcąc zapewnić pracę beż przekłamań niezależnie od opóźnień elementów i bez potrzeby korygowania opóźnień, należy syntezę bloku UBZCz przeprowadzić jako syntezę układu sekwencyjnego. Wynika to z następujących przesłanek.

Dla projektowanego bloku UBZCz przed jego syntezą nie można wiedzieć jakie będą opóźnienia na drodze od wejścia x do wyjścia Z, a jakie od wejścia sygnałów czasowych: t_1 i t_2 do wyjścia Z.

Zatem w programie pracy układu należy przyjąć różne rodzaje kolejności zmiany sygnału x i sygnału t_1 przy przejściu: $t_1t_2x = 000 \rightarrow 101$, a także przyjąć różne rodzaje kolejności zmiany sygnału x i sygnału t_2 przy przejściu: $t_1t_2x = 001 \rightarrow 010$. Uwzględniając te rodzaje pracy układu w jego programie pracy są takie stany wejść: $xt_1t_2 = 100$ oraz $xt_1t_2 = 000$, w których wyjście Z może przyjmować różne wartości. Zatem w tej wersji układ należy zaprojektować, jako sekwencyjny.

Przykład 11.2

Należy przeprowadzić syntezę układu opisanego wykresem czasowym podanym na rys. 11-8a.



Rys. 11-8

a)

q_1q_2	t = 0	t = 1	Z
00	(1)	2	0
01	3	(2)	0
11	(3)	4	1
10	1	(4)	0

b)

q_1q_2	t = 0	t = 1
00	00	01
01	11	01
11	11	10
10	00	10

Q_1Q_2

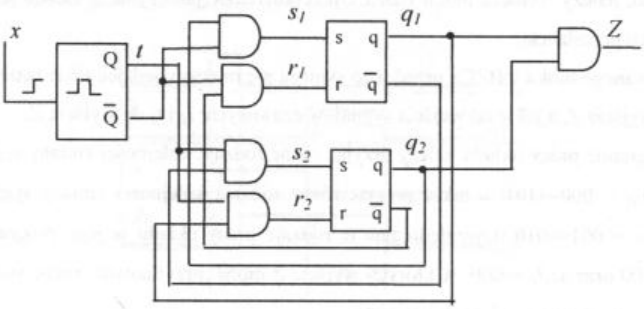
c)

$$Z = q_1 q_2$$
$$s_1 = q_2 \bar{t}, \quad r_1 = \overline{q_2}\,\bar{t},$$
$$s_2 = \overline{q_1}\, t \quad r_2 = q_1 t.$$

Rys. 11-9

Do realizacji opóźnień zastosowano element czasowy wyzwalany narastającym zboczem sygnału x i posiadający wyjście oznaczone sygnałem t. Z rysunku 11-8b wynika, że zmiany sygnału Z są jednoznacznie określone zmianami sygnału t. Zatem w dalszych etapach syntezy można pominąć sygnał x.

Z wykresu czasowego podanego na rys. 11-8b wynika, że rozpatrywany układ jest układem sekwencyjnym. Pierwotna siatka programu, siatka stanów wewnętrznych, wyrażenia dla funkcji wyjścia i funkcji wzbudzeń przerzutników sr użytych do realizacji sygnałów q_1 i q_2 przedstawiono na rysunkach odpowiednio: 11-9a, 11-9b oraz 11-9c. Schemat logiczny rozwiązania przedstawia rysunek 11-10.

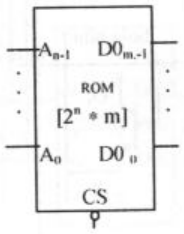


Rys. 11-10

Przykład syntezy układu z zależnościami czasowymi realizowanego w wersji układu mikroprogramowanego przedstawiony jest w rozdziale 13 (przykład 13.4).

12. PAMIĘCI STAŁE

Pamięci stałe ROM (ang. Read Only Memory) stanowią zbiór komórek, w których przechowywany jest jeden bit informacji (0 lub 1) lub m bitów, czyli słowo. Zawartość jej można odczytywać po podaniu adresu (zwykle w kodzie naturalnym dwójkowym). Zawartość ta jest zachowana po wyłączeniu napięcia zasilania pamięci. Pojemność pamięci wyrażona jest jako $[2^n * m]$ gdzie: n - liczba wejść adresowych, 2^n - liczba komórek m-bitowych, m - długość słowa. Rys. 12-1 przedstawia symbol i tablicę zawartości pamięci ROM.



Adres pamięci	Zawartość pamięci
$A_{n-1}...A_0$	$D0_{m-1}...D0_0$
0 ... 0	0 ... 1
0 ... 1	0 ... 0
...	...
1 ...1	0 ... 1

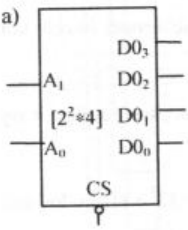
Rys. 12-1

Zawartość pamięci stałej może być ustalana:

- przez producenta (ROM); użytkownik nie może jej zmienić,
- przez użytkownika jednokrotnie (PROM - ang. Programmable ROM),
- przez użytkownika wielokrotnie w procesie programowania (EPROM - ang. Erasable PROM lub E^2PROM - ang. Electrically EPROM).

Dla pamięci stałej o pojemności $[2^2 * 4]$ na rys. 12-2 przedstawiono jej symbol (a) oraz tablicę zawartości (b). Struktura wewnętrzna tej pamięci przedstawiona jest na rys. 12-3.

a)



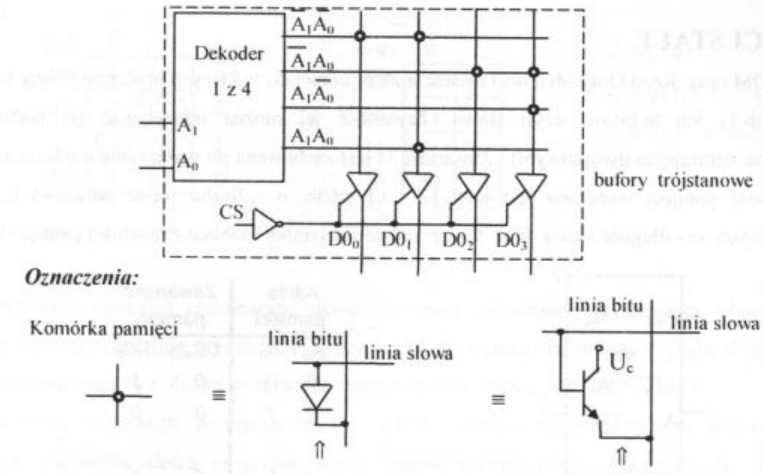
A_1, A_0 - wej. adresowe
$D0_3$÷$D0_0$ - wyj. informacyjne
CS - wej. strobujące

b)

A_1A_0	$D0_3$	$D0_2$	$D0_1$	$D0_0$
0 0	1	1	0	0
0 1	0	0	1	1
1 0	0	0	0	1
1 1	0	1	0	0

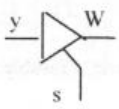
Rys. 12-2

W pamięciach PROM w miejscu wskazanym na rys. 12-3 strzałką ⇑ występuje bezpiecznik. Przepalenie bezpiecznika pozwala na jednorazową zmianę zawartości pamięci.



Rys. 12-3

Wyjścia pamięci wyprowadzone są z buforów trójstanowych (bramek trójstanowych). Opis działania bramki trójstanowej podaje rys. 12-4. „Stan trzeci" jest takim stanem, dla którego wyjście bramki zachowuje się tak, jakby było odłączone od bramki (stan wysokiej impedancji).



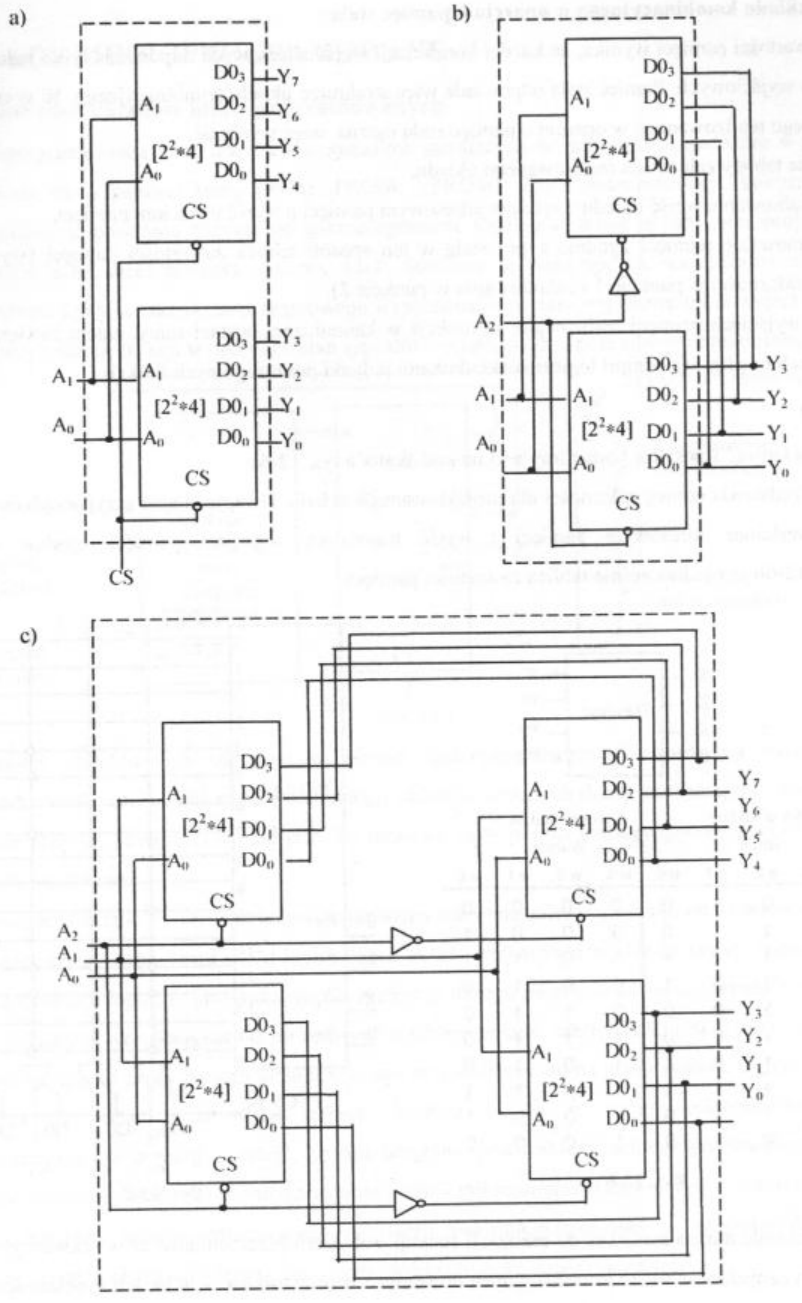
s	y	W
0	0	stan trzeci (stan wysokiej impedancji)
0	1	stan trzeci (stan wysokiej impedancji)
1	0	0
1	1	1

Rys. 12-4

Dzięki zastosowaniu bramek trójstanowych w buforach wyjściowych można budować bloki pamięci o zwiększonej liczbie komórek i zwiększonej długości słowa. W blokach o zwiększonej liczbie komórek dzięki zastosowaniu tych bramek możliwe jest zwieranie wyjść pamięci.

Rys. 12-5 przedstawia przykład budowy większych bloków pamięci stałej w oparciu o pamięci z wyjściami trójstanowymi o mniejszych rozmiarach. Przykład pamięci o:

- zwiększonej długości słowa przedstawia rys. 12-5a - pamięć o rozmiarach: $[2^2 * 8]$ zbudowana w oparciu o pamięć o rozmiarach $[2^2 * 4]$,
- zwiększonej liczbie komórek przedstawia rys. 12-5b - pamięć o rozmiarach: $[2^3 * 4]$ zbudowana w oparciu o pamięć o rozmiarach $[2^2 * 4]$,
- zwiększonej długości słowa i zwiększonej liczbie komórek rys. 12-5c - pamięć o rozmiarach: $[2^3 * 8]$ zbudowana w oparciu pamięć o rozmiarach $[2^2 * 4]$.



Rys. 12-5

Realizacja układu kombinacyjnego w oparciu o pamięć stałą

Z tablicy zawartości pamięci wynika, że każdej kombinacji wejść adresowych odpowiada tylko jedna kombinacja sygnałów wyjściowych. Pamięć stała odpowiada więc strukturze układu kombinacyjnego. W syntezie układu kombinacyjnego realizowanego w oparciu o pamięć stałą można więc wyróżnić:

1. Określenie tablicy zależności realizowanego układu,
2. Przyporządkowanie wejść układu wejściom adresowym pamięci a wyjść wyjściom pamięci,
3. Zaprogramowanie pamięci zgodnie z powstałą w ten sposób tablicą zawartości pamięci (wynikającą z tablicy zależności w punkcie 1 i zakodowania w punkcie 2).

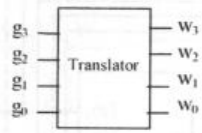
Uwaga. Na wyjściach pamięci realizowane są funkcje w kanonicznej postaci sumy, zatem zawierają HSd przy przejściach między sąsiednimi logicznie składnikami jedynki poszczególnych funkcji.

Przykład 12.1

Należy zaprojektować translator kodu Gray'a+3 na kod Watts'a rys. 12-6a.

Rys. 12-6b przedstawia tablicę zależności dla projektowanego układu. Uwzględniając przyporządkowanie wejść translatora wejściom adresowym pamięci i wyjść translatora wyjściom pamięci zgodnie z rys. 12-7, tablica z rys. 12-6b jest jednocześnie tablicą zawartości pamięci.

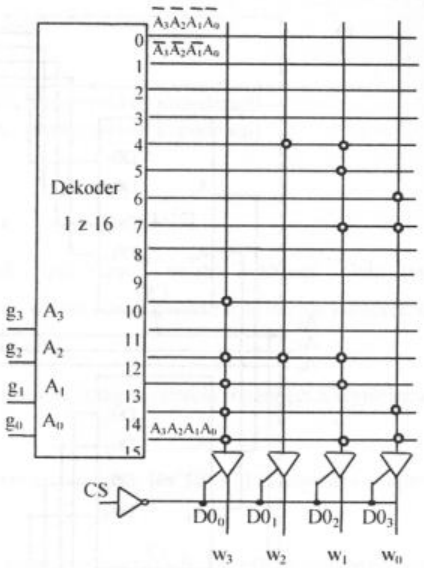
a)



b)

Cyfra w kodzie Gray'a +3				Cyfra w kodzie Watts'a			
g 3	g 2	g 1	g 0	w 3	w 2	w 1	w 0
0	0	1	0	0	0	0	0
0	1	1	0	0	0	0	1
0	1	1	1	0	0	1	1
0	1	0	1	0	0	1	0
0	1	0	0	0	1	1	0
1	1	0	0	1	1	1	0
1	1	0	1	1	0	1	0
1	1	1	1	1	0	1	1
1	1	1	0	1	0	0	1
1	0	1	0	1	0	0	0

Rys. 12-6



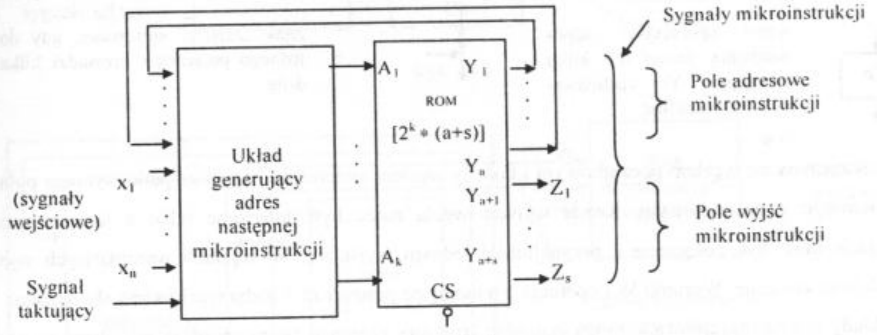
Rys. 12-7

Pamięci stałe można stosować do realizacji funkcji wzbudzeń przerzutników *sr* w sekwencyjnych układach asynchronicznych oraz wejść informacyjnych dowolnych przerzutników w układach synchronicznych. Realizacja układów synchronicznych z zastosowaniem pamięci stałych w wersji mikroprogramowanej jest opisana w osobnym rozdziale.

13. UKŁADY MIKROPROGRAMOWANE

13.1. Ogólna idea układów mikroprogramowanych

Istotą mikroprogramowania jest organizacja sygnałów sterujących w mikroinstrukcje, które w postaci słów przechowywane są w pamięci stałej (ROM, PROM, EPROM). Zbiór mikroinstrukcji, których wykonanie zapewnia realizację programu nazywa się mikroprogramem. Ogólną strukturę układu mikroprogramowanego można wyrazić schematem podanym na rys. 13-1. Struktura ta wskazuje na sekwencyjny synchroniczny charakter układu, który w takt sygnału zegarowego wypracowuje sygnały wyjściowe (pole wyjść) i modyfikuje adres następnej mikroinstrukcji w oparciu o stan sygnałów wejściowych i pole adresowe mikroinstrukcji.



Rys. 13-1

Metoda projektowania układów w postaci mikroprogramowanej[1] stosowana była początkowo w maszynach cyfrowych. Szybki rozwój technologii układów scalonych dużej skali integracji spowodował, że pamięci stałe stały się powszechnie dostępne. To sprawiło, że w postaci mikroprogramowanej realizuje się dowolne układy sterowania.

Realizacja układu w wersji mikroprogramowanej ma wiele zalet w stosunku do układów realizowanych w tzw. „logice szytej" (ang. Hard wired logic). Przez układy realizowane w „logice szytej" rozumie się układy posiadające strukturę logiczną indywidualną dla realizacji danego programu automatu. Synteza ich przeprowadzana jest opisanymi wcześniej metodami opartymi o tablicę przejść, tablicę kolejności łączeń lub w inny sposób. Zmiana programu pracy takiego układu wymaga przeprowadzenia na nowo syntezy logicznej i układowej jak również na nowo zmontowania układu. Synteza układu w wersji mikroprogramowanej[2] jest zazwyczaj prostsza niż w wersji „szytej". Zmiana programu pracy układu mikroprogramowanego nie wymaga zmiany jego struktury układowej i wymaga jedynie zmiany mikroprogramu zapisanego w pamięci stałej. Mniejsza liczba połączeń układu realizowanego w wersji mikroprogramowanej w stosunku do rozwiązania w strukturze „szytej" zapewnia większą niezawodność układu mikroprogramowanego.

---

[1] Opracowana w 1951r przez M.V. Wilkesa

[2] Uwzględniona w programach dydaktycznych zawartych w pikach: Mikr-nau.zip oraz Mikr-syn.zip na witrynie internetowej: http://zmitac.iinf.polsl.gliwice.pl

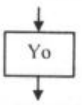
Najczęściej stosowanym sposobem opisu pracy układu mikroprogramowanego jest graf przejść lub sieć działań. Graf przejść był omawiany wcześniej.

Zasady opisu układu w postaci sieci działań

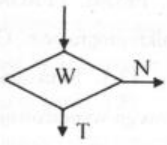
Sieć działań jest grafem spójnym, zawierającym następujące typy wierzchołków (węzłów, bloków):

– węzeł początkowy, uruchamiający program, nie mający poprzednika.

– węzeł końcowy, kończący program, nie mający następnika.



– węzeł operacyjny, odpowiadający jednej lub kilku operacjom Yo realizowanym jednocześnie.



– węzeł warunkowy, w którym wpisuje się warunek W jaki jest niezbędny dla realizacji „rozejścia się" w algorytmie programu układu, zależnie od spełnienia (T) lub niespełnienia (N) warunku W.

– znak „zejście" stosowany, gdy do jednego połączenia prowadzi kilka dróg

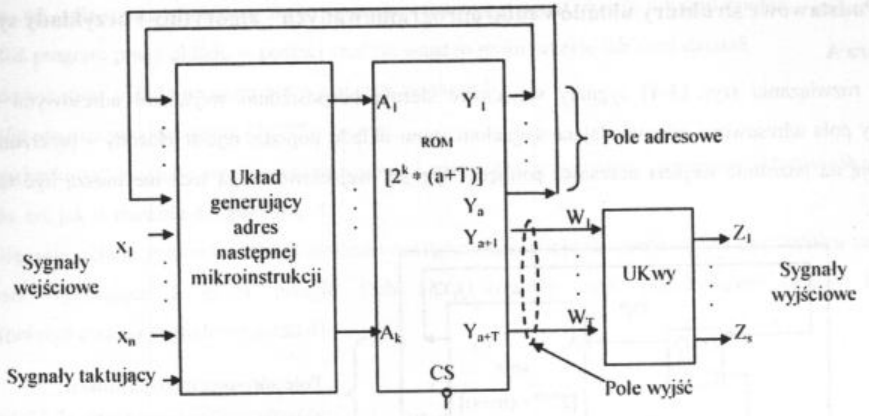
Sieć działań zaczyna się węzłem początkowym i kończy węzłem końcowym, z tym że pętla wymaga połączenia węzła końcowego z początkowym. Każde wyjście węzła może być połączone tylko z jednym wejściem. Każde wejście musi być połączone z przynajmniej jednym wyjściem. W węzłach warunkowych występują dwuwartościowe zmienne. Warunki W i operacje Yo mogą się powtarzać. Liczba węzłów jest skończona.

Układy mikroprogramowane mogą posiadać struktury różniące się w zasadzie sposobem generowania sygnałów wyjściowych i adresu następnej mikroinstrukcji.

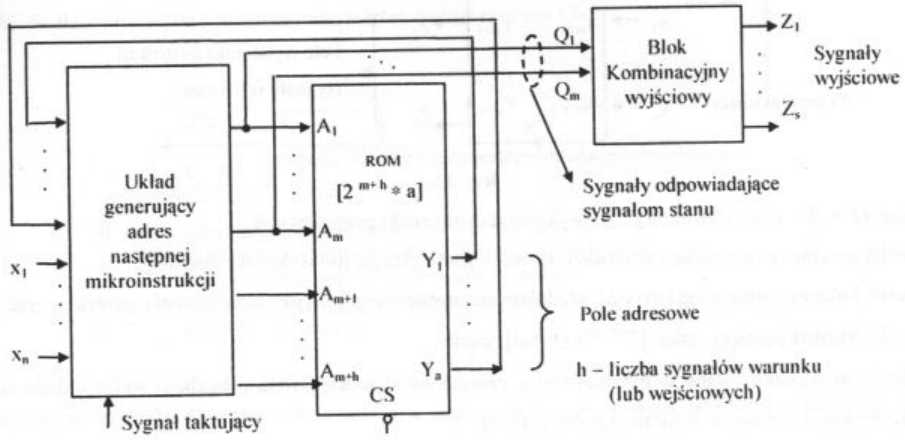
Sposoby generowania sygnałów wyjściowych

1. Sygnałom wyjściowym odpowiadają bezpośrednio sygnały pola adresowego (rys. 13-1).
2. Pole wyjść mikroinstrukcji zawiera sygnały : W_1, .. W_T kodujące stany wyjść (gdzie: $T < s$, s – liczba sygnałów wyjściowych układu), a Układ Kombinacyjny wyjściowy (Ukwy) wypracowuje (dekoduje) sygnały wyjściowe. Takie rozwiązanie (rys. 13-2) jest stosowane wtedy, kiedy liczba stanów wyjść układu jest mniejsza niż 2^s i nie większa niż 2^T.
3. Sygnały wyjściowe są realizowane w oparciu o sygnały odpowiadające sygnałom stanu układu w klasycznym BKwy (rys. 13-3). To rozwiązanie można traktować jako szczególny przypadek sposobu z punktu 2, gdy sygnały stanu Q_1, .. Q_m odpowiadają sygnałom W_1, .. W_T. Stosowane jest wtedy, kiedy występuje stosunkowo prosta zależność sygnałów wyjściowych od sygnałów stanu. W szczególnym przypadku, gdy stany układu zakodowane są w sposób zgodny ze stanem wyjść, BKwy sprowadza się do prostych wyprowadzeń: $Z_1 = Q_1$, ..., $Z_s = Q_s$.

Rozwiązania w wersji 2 i 3 pozwalają zmniejszyć rozmiary potrzebnej pamięci stałej w porównaniu z rozmiarami pamięci w strukturze wg wersji 1, poprzez zmniejszenie długości słowa mikroinstrukcji. Pole adresowe posiada mniej bitów (wersja 2) lub jest zbędne (wersja 3).



Rys. 13-2



Rys. 13-3

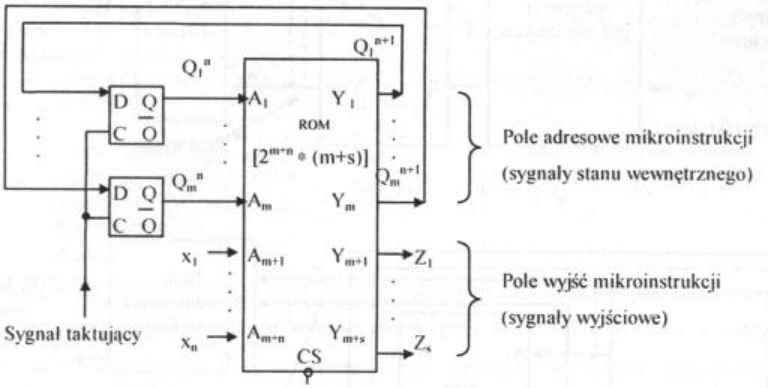
Sposoby generowania adresu następnej mikroinstrukcji

Wybrane trzy (spośród wielu) sposoby są dalej zaprezentowane w postaci struktur: A, B i C. Struktura A stanowi najprostszy sposób generowania adresu następnej mikroinstrukcji. W niej liczba wejść adresowych pamięci stanowi sumę liczby sygnałów stanu i liczby sygnałów wejściowych układu. Struktury B i C pozwalają zmniejszyć rozmiary potrzebnej pamięci stałej w stosunku do struktury A poprzez zmniejszenie liczby wejść adresowych pamięci stałej. W strukturze B w miejsce wszystkich wejść układu na wejścia adresowe pamięci wprowadzane są tylko sygnały (jeden lub kilka) stanowiące warunek przejścia w danym stanie. W strukturze C liczba wejść adresowych pamięci ogranicza się do liczby sygnałów stanu układu.

13.2. Podstawowe struktury układów mikroprogramowanych - algorytmy i przykłady syntezy

Struktura A

W tym rozwiązaniu (rys. 13-4) sygnały wejściowe sterują bezpośrednio wejściami adresowymi pamięci, a sygnały pola adresowego odpowiadające sygnałom stanu układu poprzez rejestr złożony z przerzutników D oddziałują na pozostałe wejścia adresowe pamięci. Sygnały wejściowe mogą lecz nie muszą być taktowane zegarem.



Rys. 13-4

Algorytm A1 - dla syntezy Struktury A z wykorzystaniem siatki przejść/wyjść

1. Określić program pracy układu w postaci: słownej, grafu przejść lub wykresu czasowego.
2. Określić kolejno: siatkę przejść/wyjść, zredukowaną siatkę przejść/wyjść, binarną siatkę przejść/wyjść.
3. Określić rozmiar pamięci stałej: $[2^{(m+n)} * (m+s)]$, gdzie:
   - liczba m sygnałów stanu w binarnej siatce przejść/wyjść powiększona o liczbę n wejść układu stanowi liczbę wejść adresowych pamięci stałej: $(m+n)$,
   - liczba sygnałów stanu m powiększona o liczbę wyjść s wskazuje długość słowa pamięci: $(m+s)$.
4. Przyporządkować sygnały: wejściowe, wyjściowe i stanu układu sygnałom elementów składowych struktury A zgodnie z ogólnym schematem podanym na rys. 13-4, a mianowicie:
   - sygnałom wejściowym przerzutników D tworzącym rejestr - sygnały pola adresowego, czyli sygnały stanu w $n+1$–szym przedziale czasu (Q^{n+1}),
   - wejściom adresowym pamięci - sygnały wejściowe układu i sygnały wyjściowe rejestru, czyli sygnały stanu układu w n- tym przedziale czasu (Q^n),
   - sygnałom wyjściowym układu sygnały pola wyjść pamięci.
5. Rozwiązanie układu stanowi schemat struktury układu z uwzględnieniem wyników punktu 3 i 4 oraz tablica zawartości pamięci wynikająca z binarnej siatki przejść/wyjść (punkt 2) i przyporządkowania sygnałów w punkcie 4.

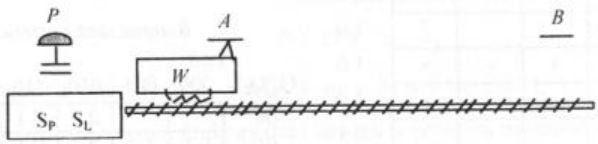
Rozwiązanie można też podać bezpośrednio w oparciu o zredukowany i zakodowany graf przejść.

Algorytm A2 - dla syntezy Struktury A bezpośrednio w oparciu o graf przejść lub sieć działań

1. Określić program pracy układu w postaci zredukowanego grafu przejść lub sieci działań.
2. Zakodować stany układu występujące w opisie w punkcie 1.
3. Określić rozmiar pamięci stałej: $[2^{(m+n)} * (m+s)]$.
4. Przyporządkować sygnały: wejściowe, wyjściowe i stanu układu sygnałom elementów składowych struktury układu, tak jak w punkcie 4 ***algorytmu A1***.
5. Rozwiązanie układu stanowi schemat struktury uwzględniający wyniki punktu 2 i 3 oraz tablica zawartości pamięci wynikająca z grafu przejść (lub sieci działań) oraz zakodowania stanów (punkt 2) i przyporządkowania sygnałów (punkt 4).

Przykład 13.1 - *synteza układu o strukturze A*

Należy przeprowadzić syntezę logiczną układu sterowania ruchem wózka *W*. Położenie wózka kontrolują czujniki *A* i *B* rozmieszczone przestrzennie, tak jak podaje rysunek 13-5.



S_P - stycznik prawych obrotów silnika powodujący ruch wózka w prawo,
S_L - stycznik lewych obrotów silnika powodujący ruch wózka w lewo
$a = 1$ - wózek znajduje się w polu działania czujnika *A*,
$b = 1$ - wózek znajduje się w polu działania czujnika *B*,
$p = 1$ - wciśnięty przycisk *P*.

Rys. 13-5

Po naciśnięciu przycisku *P* wózek znajdujący się w położeniu *A* (a = 1) powinien ruszyć w kierunku położenia *B*. Po dojechaniu do *B* (b = 1) powinien zawrócić a następnie zatrzymać się w położeniu *A*.

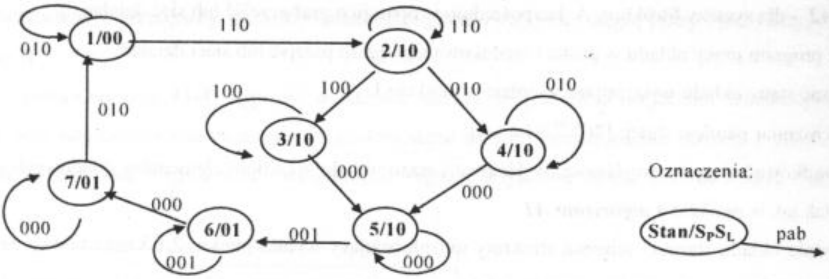
Dla sterowania ruchem wózka należy wypracować sygnały:

rodzaj pracy	S_P S_L
ruch w prawo	1 0
ruch w lewo	0 1
zatrzymanie	0 0

Założenia:

1. Przycisk *P* po uruchomieniu wózka zostaje zwolniony przed dojechaniem wózka do położenia *B*.
   Nie może on być użyty ponownie przed powrotem wózka do położenia *A*.
2. Należy uwzględnić różną kolejność pojedynczych zmian sygnałów *p* oraz *a* przy przejściu: $(110 \Rightarrow 000)_{pab}$.
3. Wózek nie może się znaleźć na prawo od czujnika B ani na lewo od czujnika A.

Pracę układu opisuje graf przejść podany na rys. 13-6.



Rys. 13-6

Siatka przejść/wyjść

S^n \ *pab*	000	001	010	110	100	S_PS_L
1	-	-	1	2	-	0 0
2	-	-	4	2	3	1 0
3	5	-	-	-	3	1 0
4	5	-	4	-	-	1 0
5	5	6	-	-	-	1 0
6	7	6	-	-	-	0 1
7	7	-	1	-	-	0 1

S^{n+1}

Zredukowana siatka przejść/wyjść

S^n \ *pab*	000	001	010	110	100	S_PS_L
1	-	-	1	2	-	0 0
2,3,4,5	5	6	4	2	3	1 0
6,7	7	6	1	-		0 1

S^{n+1}

Binarna siatka przejść/wyjść

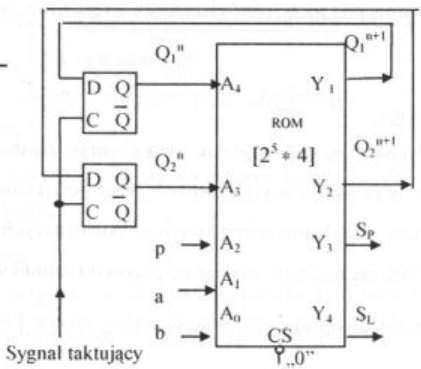
$Q_1^nQ_2^n$ \ *pab*	000	001	010	110	100	S_PS_L
00			0 0	0 1		0 0
01	0 1	1 1	0 1	0 1	0 1	1 0
11	1 1	1 1	0 0			0 1
10						

$Q_1^{n+1}Q_2^{n+1}$

Rys. 13-7

Z siatki binarnej przejść/wyjść (rys. 13-7) wynikają wartości: m = 2, n = 3, s = 2, a z nich potrzebne rozmiary pamięci stałej: $[(2^{m+n}) * (m + s)] = [2^5 * 4]$ oraz schemat struktury (rys. 13-8). Z przyporządkowania sygnałów i siatki binarnej wynika tablica zawartości pamięci podana na rys. 13-8.

Q_1^n	Q_2^n	p	a	b	Q_1^{n+1}	Q_2^{n+1}	S_P	S_L
A_4	A_3	A_2	A_1	A_0	Y_1	Y_2	Y_3	Y_4
0	0	0	1	0	0	0	0	0
0	0	1	1	0	0	1	0	0
0	1	0	0	0	0	1	1	0
0	1	0	0	1	1	1	1	0
0	1	0	1	0	0	1	1	0
0	1	1	1	0	0	1	1	0
0	1	1	0	0	0	1	1	0
1	1	0	0	0	1	1	0	1
1	1	0	0	1	1	1	0	1
1	1	0	1	0	0	0	0	1



Rys. 13-8

Rozwiązanie można też podać bezpośrednio w oparciu o zredukowany graf przejść lub sieć działań (rys. 13-9).

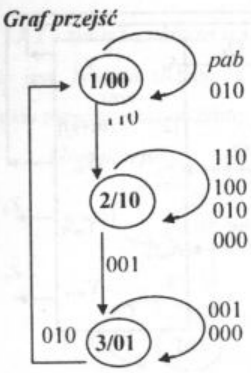
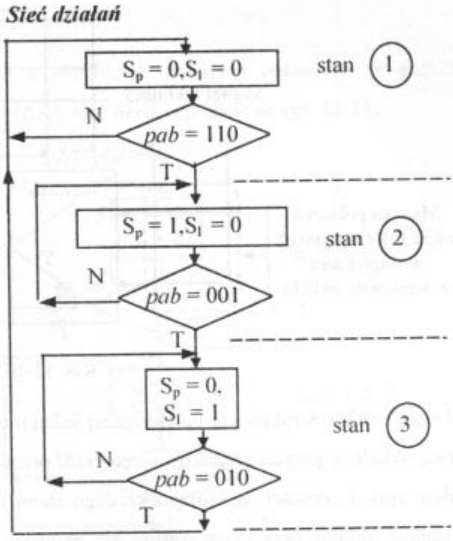


Tabela kodowania

Stan	Q_1 Q_2
1	0 0
2	0 1
3	1 1



Rys. 13-9

Z grafu lub sieci działań wynikają wartości: m = 2, n = 3, s = 2, a z nich: rozmiary pamięci: $[(2^{m+n}) * (m+s)] = [2^5 * 4]$ oraz schemat struktury (taki sam jak w podanym wcześniej rozwiązaniu).

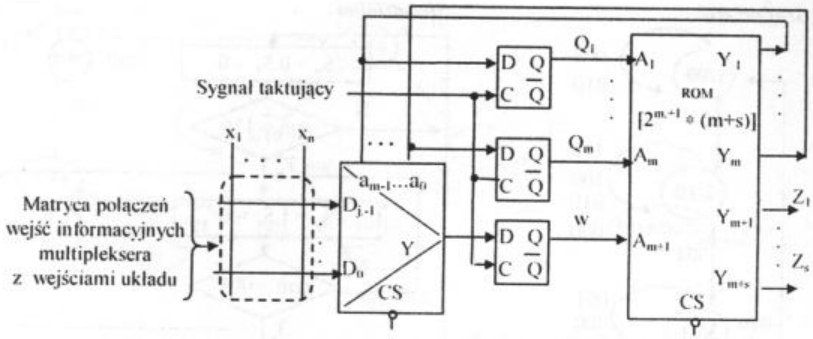
Z przyporządkowania sygnałów zgodnie z rysunkiem określającym strukturę oraz z przejść uwzględnionych na grafie lub sieci działań, wynika tablica zawartości pamięci (taka jaką przedstawiono na rys. 13-8 w wyniku wcześniej omówionego sposobu syntezy).

W rozwiązaniu wg rys. 13-8 kodowanie stanów układu nie uwzględnia stanu wyjść. Zmieniając go na zgodne ze stanem wyjść można zmniejszyć rozmiary potrzebnej pamięci, tak jak to opisano w przykładzie 13.2.

Struktura B

Oprócz pamięci stałej zawiera ona multiplekser warunku (jeden lub kilka) i rejestr z przerzutników D. Na wejścia adresowe pamięci oprócz sygnałów stanu, przekazywana jest wartość (wybranych za pomocą multiplekserów) tylko tych sygnałów wejściowych, które stanowią warunek przejścia w danym stanie układu. Zastosowanie multipleksera warunku ma na celu zmniejszenie rozmiarów pamięci w porównaniu ze strukturą A poprzez zmniejszenie liczby wejść adresowych o wartość *n-hw*, gdzie: *n* – liczba wejść, *hw* – liczba multiplekserów warunku. Zastosowanie tej struktury nie zawsze jednak pozwala zmniejszyć rozmiary pamięci. Wymaga ona bowiem takiego opisu pracy układu, aby w każdym stanie kontrolować nie więcej niż *hw* wejść układu. Dla niektórych układów wymaga to zwiększenia liczby stanów układu, które pociąga za sobą zwiększenia liczby sygnałów stanu, a przez to zwiększenie liczby bitów pola adresowego wyjść pamięci. Dalej rozważana jest tylko struktura B z jednym[1] multiplekserem warunku (rys. 13-10).

[1] Przykłady struktur z kilkoma multiplekserami warunku podano w ćwiczeniu 14 [6].



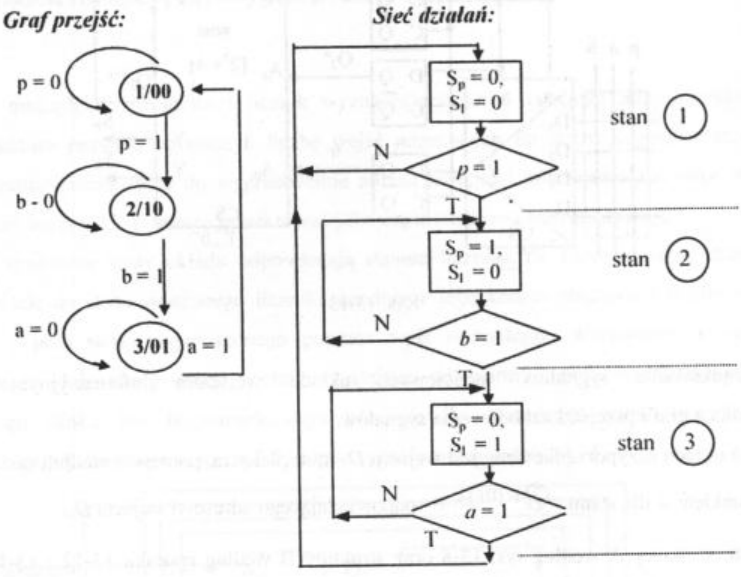
Rys. 13-10

Algorytm A3 - dla syntezy Struktury B zawierającej jeden multiplekser warunku

1. Opisać pracę układu w postaci zredukowanego grafu przejść lub sieci działań.
2. Przekształcić opis do postaci, w której z każdego stanu jest warunkowe przejście co najwyżej do jednego z dwóch stanów układu, przy czym warunkiem przejścia jest stan jednego tylko sygnału wejściowego lub przejście bezwarunkowe, a następnie zakodować stany układu. Sposób kodowania stanów może mieć wpływ na rozmiary multipleksera.
3. Określić rozmiar pamięci stałej jako: $[(2^{m+1}) * (m+s)]$, gdzie:
   - liczba *m* sygnałów stanu w grafie przejść lub sieci działań powiększona o 1 (o jedno wyjście multipleksera warunku) stanowi liczbę wejść adresowych pamięci stałej: (m+ 1).
   - liczba sygnałów stanu *m* powiększona o liczbę wyjść *s* wskazuje długość słowa pamięci: (m+ s).
4. Przyporządkować sygnały: wejściowe i wyjściowe układu sygnałom elementów składowych struktury B zgodnie ze schematem podanym na rys. 13-10, a mianowicie:
   - sygnałom wejściowym przerzutników D tworzącym rejestr - sygnał wejściowy multipleksera oraz sygnały pola adresowego wyjść pamięci, czyli sygnały stanu układu (Q^{n+1}),
   - wejściom adresowym pamięci - wyjścia rejestru, czyli sygnały stanu Q^n i warunek przejścia *w*,
   - wyjściom pamięci sygnały wyjściowe układu i sygnały stanu w przedziale n+1 (Q^{n+1}),
   - wejściom informacyjnym multipleksera wejścia układu zgodnie z zasadą: sygnał wejściowy x_i przyporządkowany jest wejściom informacyjnym multipleksera o adresach odpowiadających kodom stanów układu, w których według grafu lub sieci działań sygnał x_i jest warunkiem *w*,
   - wejściom adresowym multipleksera sygnały pola adresowego niektóre lub wszystkie (tylko te, które są niezbędne dla jednoznacznego rozróżnienia sygnałów będących warunkiem *w*).
5. Rozwiązanie układu stanowi schemat struktury uwzględniający wyniki punktu 3 i 4 oraz tablica zawartości pamięci wynikająca z grafu przejść i zakodowania stanów (punkt 1 i 2) oraz przyporządkowania sygnałów (punkt 4).

Przykład 13.2 - synteza układu o strukturze B

Rozważana jest synteza układu o strukturze B i warunkach działania opisanych w przykładzie 13.1. Dla określania rozwiązania potrzebny jest graf przejść lub sieć działań podane na rys. 13-11.



Rys. 13-11

Na tym rysunku (w odróżnieniu od rys. 13-9) graf i sieć działań wyraźnie wskazują sygnał zwany warunkiem *w*, od wartości którego zależy zachowanie się układu w danym stanie. Dla zakodowania stanów, takiego jak dla struktury A wynika rozwiązanie podane w postaci tablicy zawartości pamięci (rys. 13-12) oraz schematu logicznego (rys. 13-13).

Tabela kodowania

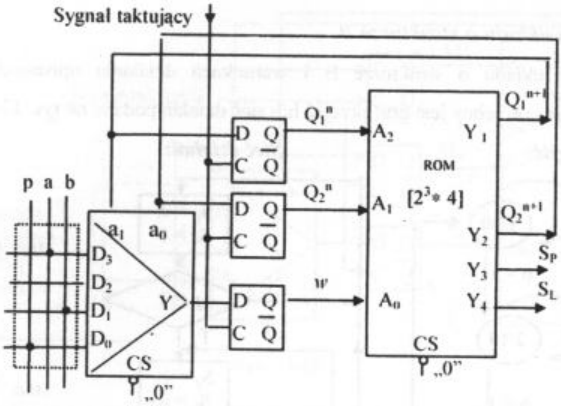
Stan	Q_1 Q_2
1	0 0
2	0 1
3	1 1

Tablica zawartości pamięci

	Q_1^n	Q_2^n	W	Q_1^{n+1}	Q_2^{n+1}	S_P	S_L
	A_2	A_1	A_0	Y_1	Y_2	Y_3	Y_4
①	0	0	0 (p)	0	0	0	0
	0	0	1	0	1	0	0
②	0	1	0 (b)	0	1	1	0
	0	1	1	1	1	1	0
③	1	1	0 (a)	1	1	0	1
	1	1	1	0	0	0	1

Rys. 13-12

W tablicy zawartości pamięci na rys. 13-12 obok par wierszy podano stany stabilne układu, z których wynikają wartości w tych wierszach. Ułatwia to wypełnianie tablicy w oparciu o rys. 13-11.



Rys. 13-13

Przyporządkowanie sygnałów wejściowych układu wejściom informacyjnym multipleksera (rys. 13-13) wynika z grafu przejść i zakodowania sygnałów.

Np.: wejście *b* układu przyporządkowane jest wejściu D_1 multipleksera, ponieważ według rys. 13-11 sygnał *b* jest warunkiem *w* dla stanu ②≡ $(01)_{Q1\,Q2}$ odpowiadającego adresowi wejścia D_1.

W rozwiązaniach: struktury A według rys. 13-8 oraz struktury B według rysunku 13-12 i 13-13 zakodowano stany układu w sposób nie uwzględniający wartości wyjść w poszczególnych stanach.

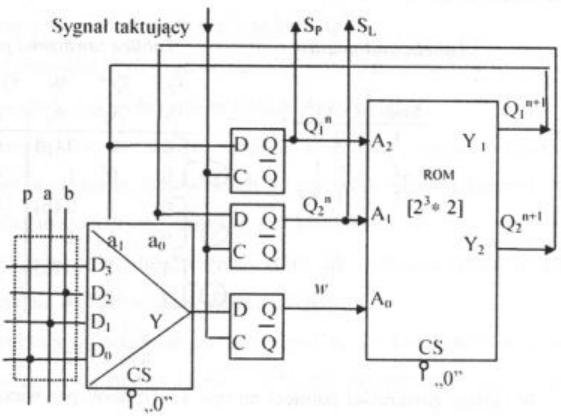
Warto zauważyć, że przy zakodowaniu stanów zgodnym ze stanami wyjść, rozwiązania mogą być znacznie prostsze, tak jak podają: tablica zawartości pamięci i schemat układu o strukturze B na rys. 13-14. Z nowego zakodowania stanów wynika oczywiście nowy sposób połączeń wejść multipleksera z wejściami układu i nowa tablica zawartości pamięci.

Tabela kodowania

Stan	Q_1 Q_2
1	0 0
2	1 0
3	0 1

Tablica zawartości pamięci

	Q_1^n	Q_2^n	W	Q_1^{n+1}	Q_2^{n+1}
	A_2	A_1	A_0	Y_1	Y_2
①	0	0	0 (p)	0	0
	0	0	1	1	0
②	1	0	0 (b)	1	0
	1	0	1	0	1
③	0	1	0 (a)	0	1
	0	1	1	0	0



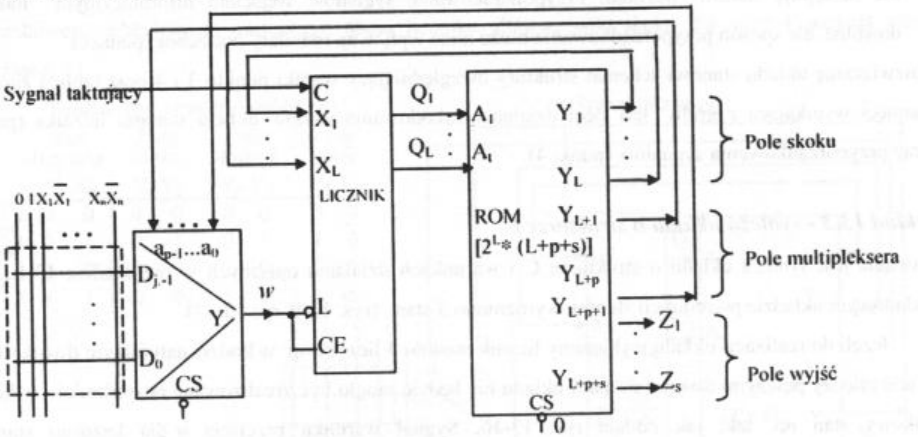
Rys. 13-14

W tym rozwiązaniu, pole wyjść w mikroinstrukcji jest zbędne, ponieważ wyjścia układu są wtedy dostępne na odpowiednich wyjściach przerzutników D. Rozmiar pamięci stałej można więc zmniejszyć do rozmiarów odpowiednio: $[2^3 * 2]$ – w przypadku struktury B. Podobnie, zmieniając kodowanie stanów w strukturze A można zmniejszyć rozmiary pamięci do: $[2^5 * 2]$.

Struktura C

Zawiera ona multiplekser warunku i licznik wyznaczający adres następnej mikroinstrukcji (rys. 13-15). Omawiana struktura pozwala ograniczyć liczbę wejść adresowych do liczby sygnałów stanu układu. Pole skoku i pole multipleksera służy do wypracowania adresu następnej mikroinstrukcji, więc w nawiązaniu do ogólnej struktury wg rys. 13-1, można je traktować jako część adresową mikroinstrukcji.

W tej strukturze stany układu odpowiadają stanom licznika. W każdym stanie układu (dla $w = 1$) następuje przejście do stanu następnego licznika zgodnie z jego kodem zliczania lub (dla $w = 0$) do stanu wynikającego z pola skoku wymuszonego poprzez wpis równoległy. Warunkowi w odpowiada sygnał wejściowy (prosty lub zanegowany) wybrany zależnie od wartości pola multipleksera. Dla realizacji bezwarunkowego skoku lub bezwarunkowego przejścia do następnego stanu licznika, na wejściach informacyjnych multipleksera powinny być dostępne sygnały odpowiednio: 0 i 1.



Opis pracy licznika: w = 1 ⇒ przejście do następnego stanu,
w = 0 ⇒ ładowanie stanu licznika dla wymuszenia rozgałęzienia (skoku)

Rys. 13-15

Algorytm A4 dla syntezy Struktury C

1. Opisać pracę układu w postaci zredukowanego grafu przejść.
2. Przekształcić graf do postaci, w której:
   - stanom układu (węzłom grafu) odpowiadają stany licznika,
   - z każdego węzła jest przejście warunkowe do następnego lub innego stanu licznika (układu),
   - warunkiem przejścia w jest jeden tylko sygnał wejściowy (prosty lub zanegowany),

- obok gałęzi przejścia zapisana jest wartość sygnału będącego warunkiem przejścia w, czyli takiego sygnału prostego lub zanegowanego, którego wartość logiczna „1" warunkuje przejście do stanu odpowiadającego następnemu stanowi licznika zgodnego z kodem liczenia, a wartość logiczna „0" warunkuje przejście do stanu odpowiadającego wymuszonemu innemu stanowi licznika.

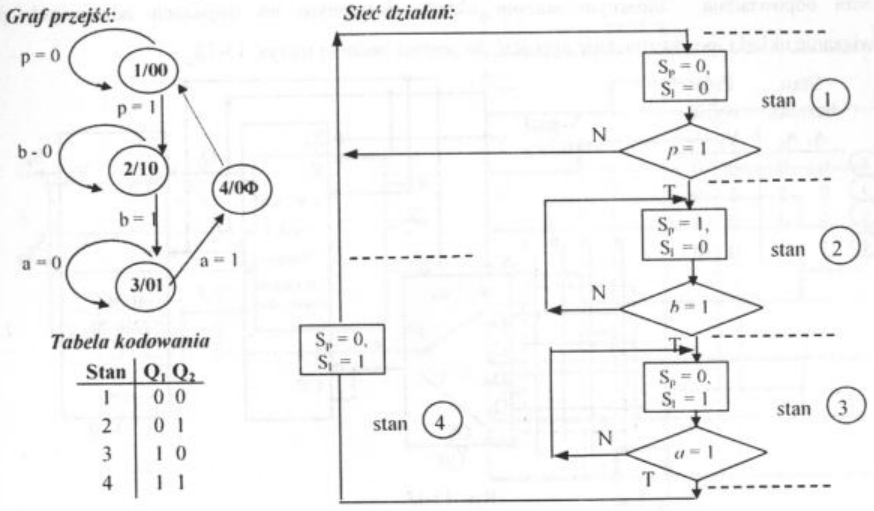
3. Określić rozmiar pamięci stałej jako $[2^L * (L + p + s)]$, gdzie: L – liczba bitów licznika, p – liczba wejść adresowych potrzebnego multipleksera, s - liczba wyjść układu.
4. Przyporządkować sygnały: wejściowe, wyjściowe i stanu układu sygnałom elementów składowych struktury C zgodnie z ogólnym schematem podanym na rys. 13-13 a mianowicie:
   - wejściom adresowym pamięci - sygnały wyjściowe licznika, czyli sygnały stanu układu (Q^n),
   - wyjściom pamięci: sygnały wyjściowe układu (pole wyjść) oraz sygnały sterujące wejściami adresowymi multipleksera (pole multipleksera) i sygnały stanu ładowania licznika (pole skoku).
   - wejściom informacyjnym multipleksera te wejścia układu proste lub zanegowane, które stanowią warunek przejścia w w jakimkolwiek stanie wg grafu lub sieci działań z punktu 2. Przy czym w=1 stanowi warunek przejścia do stanu odpowiadającego następnemu stanowi licznika zgodnie z kodem liczenia, a w=0 stanowi warunek przejścia do stanu odpowiadającego wymuszonemu (przez pole skoku), innemu niż następny, stanowi licznika. Przyporządkowanie sygnałów wejściom informacyjnym może być dowolne, ale sposób przyporządkowania może mieć wpływ na rozmiary potrzebnej pamięci.
5. Rozwiązanie układu stanowi schemat struktury uwzględniający wyniki punktu 3 i 4 oraz tablica zawartości pamięci wynikająca z grafu lub sieci działań i zakodowania stanów układu stanami licznika (punkt 2) oraz przyporządkowania sygnałów (punkt 4).

***Przykład 13.3** - synteza układu o strukturze C*

Rozważana jest synteza układu o strukturze C i warunkach działania opisanych w przykładzie 13.1. W tym przykładowym układzie po redukcji stanów wyróżniono 3 stany (rys. 13-9, rys 13-11).

Jeżeli do realizacji układu wybieramy licznik modulo 4 liczący np. w kodzie naturalnym dwójkowym, to przejście między pewnymi dwoma stanami układu nie będzie mogło być zrealizowane bezpośrednio tylko przez dodatkowy stan np. tak, jak podaje rys. 13-16. Sygnał warunku przejścia w dla każdego stanu jest zapisany obok gałęzi przejścia z tego stanu. Według rys. 13-16 przejście ze stanu 3 do 1 realizowane jest przez dodatkowy stan 4, więc wymaga dodatkowego impulsu taktującego. Przejście ze stanu 4 do stanu 1 może być realizowane jako bezwarunkowe przejście do następnego stanu licznika lub jako bezwarunkowy skok do stanu 1. Rozwiązanie przedstawia rys. 13-17.

W tablicy zawartości pamięci *Pole skoku* zawiera stan sygnałów: Q_1^{n+1} Q_2^{n+1} w przypadku, gdy sygnał warunku przejścia $w = 0$. *Pole multipleksera* zawiera adres wejścia informacyjnego multipleksera, na którym jest dostępny sygnał stanowiący warunek w w rozważanym stanie układu. *Pole wyjść* zawiera stan sygnałów wyjściowych w rozważanym stanie wewnętrznym układu.

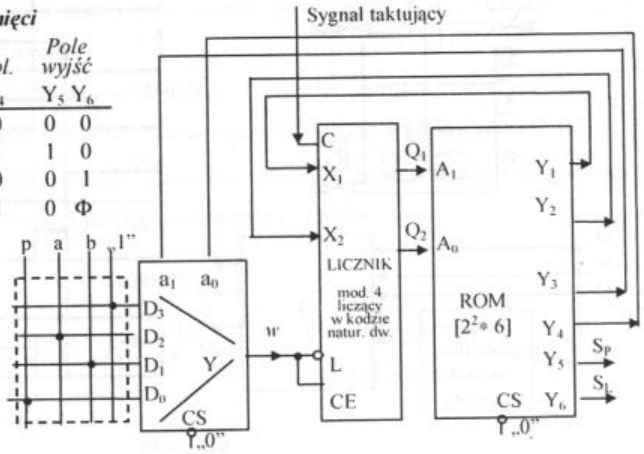


Rys. 13-16

Na rys. 13-17 podano rozwiązanie układu w postaci ogólnej struktury C z realizacją przejścia $4 \Rightarrow 1$ jako bezwarunkowego przejścia do następnego stanu licznika, dlatego pole skoku dla stanu 4 posiada wartości nieokreślone.

Tablica zawartości pamięci

	Stan licznika		Pole skoku		Pole Multipl.		Pole wyjść	
	A_1	A_0	Y_1	Y_2	Y_3	Y_4	Y_5	Y_6
(1)	0	0	0	0	0	0	0	0
(2)	0	1	0	1	0	1	1	0
(3)	1	0	1	0	1	0	0	1
(4)	1	1	Φ	Φ	1	1	0	Φ

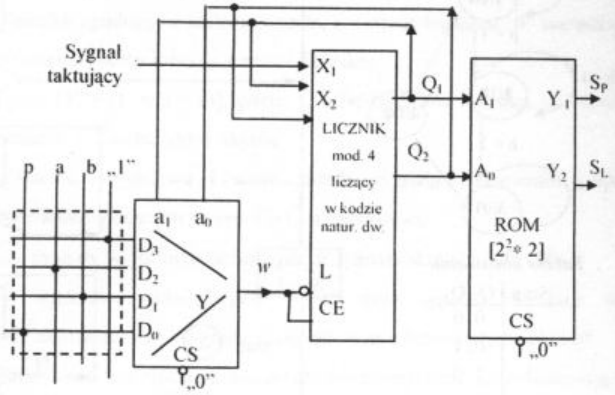


Rys. 13-17

Przyporządkowanie sygnałów stanowiących warunek w wejściom multipleksera jest dowolne, ale może ono mieć wpływ na rozmiary pamięci stałej potrzebnej do realizacji układu. Dla ilustracji tego wpływu rozważmy możliwości uproszczenia układu wynikające z przyporządkowania sygnałów podanych w matrycy połączeń na rysunku 13-17. Przeglądając wartości pól w tablicy na rys. 13-17 widać, że wartości pola skoku i pola

multipleksera odpowiadają binarnym stanom układu dostępnym na wejściach adresowych pamięci. Więc rozwiązanie układu można znacznie uprościć do postaci podanej na rys. 13-18.

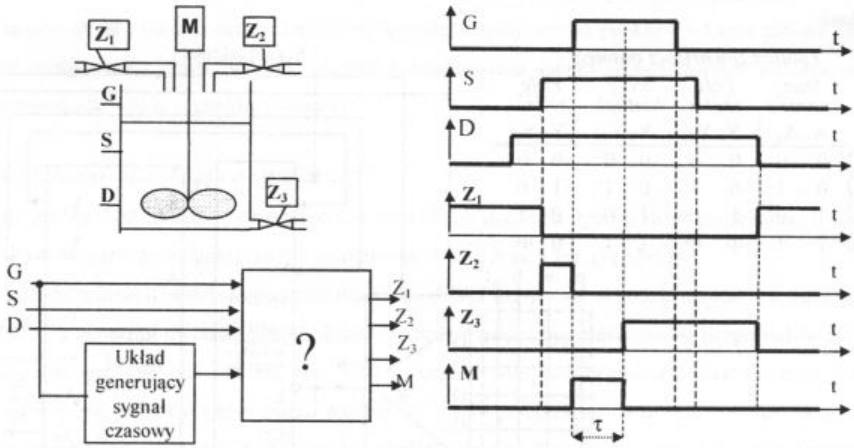
	Stan licznika		Pole wyjść	
	A_1	A_0	Y_1	Y_2
①	0	0	0	0
②	0	1	1	0
③	1	0	0	1
④	1	1	0	Φ



Rys. 13-18

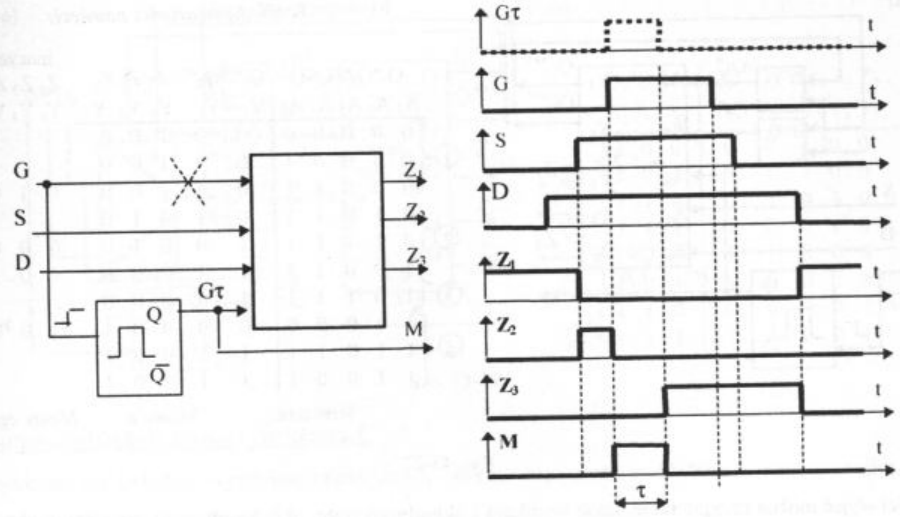
Przykład 13.4 - *synteza układu mikroprogramowanego z zależnościami czasowymi*

Należy zaprojektować układ sterujący pracą mieszalnika o działaniu opisanym wykresem czasowym podanym na rys. 13-19.



Rys. 13-19

Zbiór sygnałów wejściowych należy uzupełnić o sygnał Gτ tak, jak podaje rys. 13-20. Wtedy sygnał G jest zbędny w rozwiązaniu pozostałej części układu, ponieważ narastające jego zbocze jest reprezentowane (z dokładnością równą opóźnieniu elementu czasowego) przez sygnał Gτ, a opadające zbocze G nie wpływa na pracę układu. Można też zauważyć, że sygnał M jest równy Gτ.



Rys. 13-20

Rozwiązanie układu w postaci struktury A

Zależność sygnałów wyjściowych od wejściowych jest sekwencyjna i można ją opisać zredukowanym grafem przejść (rys. 13-21). Z grafu przejść i zakodowania stanów podanych na rys. 13-21 wynikają rozmiary potrzebnej pamięci stałej [$2^5 * 5$] ponieważ:

- liczba wejść adresowych pamięci = {liczba wejść układu (3) + liczba sygnałów stanu (2)} = 5,
- liczba bitów w słowie (liczba wyjść pamięci) = {liczba wyjść (3) + liczba sygnałów stanu (2)} = 5.

Z przyporządkowania sygnałów układu sygnałom elementów cyfrowych zgodnie ze schematem na rys. 13-22a oraz grafu przejść wynika tablica zawartości pamięci podana na rys. 13-22b.

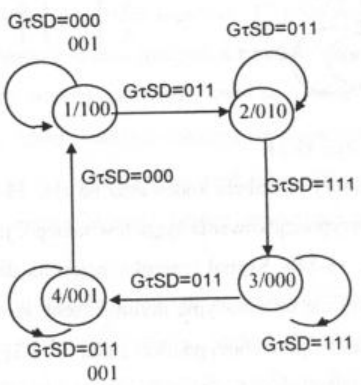
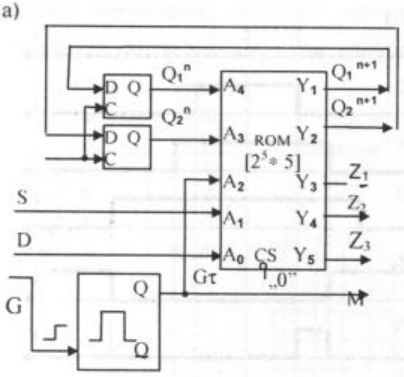


Tabela kodowania

Stan	Q_1 Q_2
1	0 0
2	0 1
3	1 0
4	1 1

Rys. 13-21

a)



b) ***Tablica zawartości pamięci:***

	$Q_1^nQ_2^nG_\tau SD$	A_4	A_3	A_2	A_1	A_0	$Q_1^{n+1}Q_2^{n+1}$	Y_1	Y_2	Z_1 Y_3	Z_2 Y_4	Z_3 Y_5	inaczej Z_1 Y_3	Z_2 Y_4	Z_3 Y_5
①	000	0	0	0	0	0	00	0	0	1	0	0			
	001	0	0	0	0	1	00	0	0	1	0	0			
	011	0	0	0	1	1	01	0	1	1	0	0	0	1	0
②	101	0	1	0	1	1	01	0	1	0	1	0			
	111	0	1	1	1	1	10	1	0	0	1	0	0	0	0
③	011	1	0	0	1	1	11	1	1	0	0	0	0	0	1
	101	1	0	1	1	1	10	1	0	0	0	0			
④	110	1	1	0	0	0	00	0	0	0	0	1	1	0	0
	111	1	1	0	1	1	11	1	1	0	0	1			
	110	1	1	0	0	1	11	1	1	0	0	1			

Struktura: Moore'a Mealy'ego

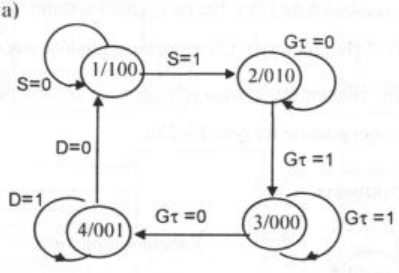
Rys. 13-22

Wartości wyjść można przyjąć takie, jakie wynikają z aktualnego stanu układu lub następnego stanu układu tak, jak podaje tablica na rys. 13-22 w kolumnie „inaczej".

Rozwiązanie układu w postaci struktury B

Działanie układu można też opisać nieco odmiennym grafem przejść podanym na rys. 13-23a, na którym obok gałęzi przejścia zapisana jest wartość tylko jednego z sygnałów wejściowych zwanego warunkiem przejścia do następnego stanu układu.

a)



b) ***Tabela kodowania***

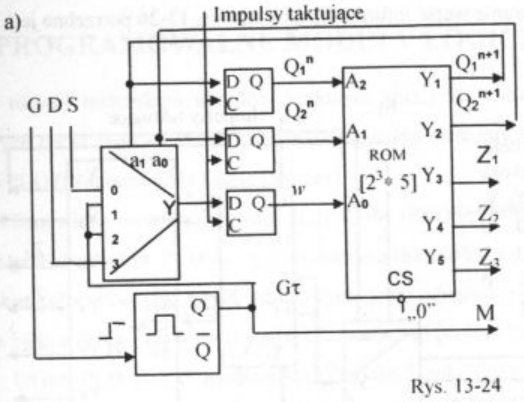
Stan	Q_1 Q_2
1	0 0
2	0 1
3	1 0
4	1 1

Rys. 13-23

Niech zakodowanie stanów jest takie, jak dla struktury A (tabela kodowania na rys. 13-23b). Z grafu przejść i zakodowania stanów zgodnym z rys 13-23 oraz przyporządkowania sygnałów takiego, jak na rysunku 13-24a, wynika tablica zawartości pamięci podana na rys. 13-24b. Sygnał warunku przejścia dotyczący rozważanego stanu odpowiada sygnałowi wprowadzonemu na wejście informacyjne multipleksera, którego adres odpowiada binarnemu kodowi rozważanego stanu układu. Rozmiary potrzebnej pamięci stałej $[2^3 * 5]$ ponieważ:

- liczba wejść adresowych pamięci = {liczba sygnałów warunku (1) + liczba sygnałów stanu (2)} = 3,
- liczba bitów w słowie (liczba wyjść pamięci) = {liczba wyjść (3) + liczba sygnałów stanu (2)} = 5.

a)



b)

$Q_1^nQ_2^nw$	$Q_1^{n+1}Q_2^{n+1}$ Z_1 Z_2 Z_3
A_2 A_1 A_0	Y_1 Y_2 Y_3 Y_4 Y_5
0 0 0	0 0 1 0 0
① 0 0 1	0 1 1 0 0
0 1 0	0 1 0 1 0
② 0 1 1	1 0 0 1 0
1 0 0	1 1 0 0 0
③ 1 0 1	1 0 0 0 0
1 1 0	0 0 0 0 1
④ 1 1 1	1 1 0 0 1

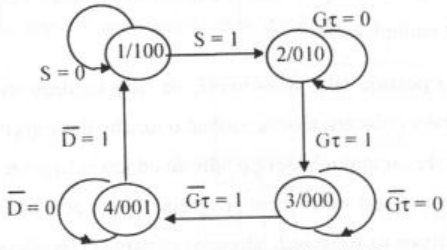
Rys. 13-24

Rozwiązania układu w postaci struktury C

W przykładowym układzie występują cztery stany. Należy więc zastosować licznik modulo 4 liczący np. w kodzie naturalnym dwójkowym:

$$(00 \rightarrow 01 \rightarrow 10 \rightarrow 11)_{Q1\,Q2}$$

a)



b)

Tabela kodowania

Stan	Q1 Q2
1	0 0
2	0 1
3	1 0
4	1 1

Rys. 13-25

Stany układu, które mogą być kolejnymi w czasie odpowiadają kolejnym stanom licznika (rys. 13-25). Sporządzenie tablicy zawartości pamięci ułatwia graf przejść (rys. 13-25a), w którym obok gałęzi przejścia zapisany jest sygnał w, którego wartość logiczna „1" warunkuje przejście do stanu odpowiadającego następnemu stanowi licznika zgodnego z kodem liczenia, a wartość logiczna „0" warunkuje przejście do stanu odpowiadającego wymuszonemu innemu stanowi licznika. Z tego grafu wynika, że warunkami przejścia są sygnały: $G_\tau, \overline{G_\tau}, S, \overline{D}$. Muszą one być dostępne na wejściach informacyjnych multipleksera, przyporządkowane na przykład tak, jak na rys. 13-26b. Uwzględniając przyporządkowanie tych i pozostałych sygnałów zgodnie z rysunkiem 13-26b można określić tablicę zawartości pamięci (rys. 13-26a). W tej tablicy dla każdego wiersza (stanu układu):

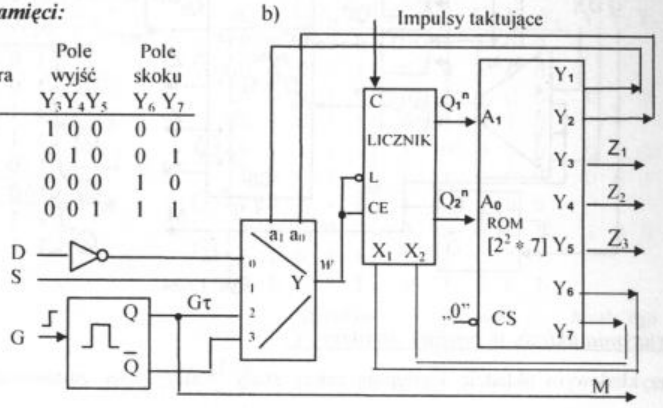
- pole multipleksera wynika z adresu wejścia informacyjnego, na którym dostępny jest sygnał będący warunkiem w w rozważanym stanie; wartość tego pola zależy również od wag poszczególnych pozycji tego pola wynikających z ich przyporządkowania wejściom adresowym multipleksera,
- pole skoku odpowiada stanowi, do którego układ przechodzi dla wartości warunku: $w = 0$.

W rozwiązaniu uwzględniającym przyporządkowanie wejść informacyjnych wg rys. 13-26 potrzebna jest pamięć o rozmiarach: $[2^2 * 7]$.

a) ***Tablica zawartości pamięci:***

	Stan licznika A_1 A_0	Pole multipleksera Y_1 Y_2	Pole wyjść Y_3 Y_4 Y_5	Pole skoku Y_6 Y_7
①	0 0	0 1	1 0 0	0 0
②	0 1	1 0	0 1 0	0 1
③	1 0	1 1	0 0 0	1 0
④	1 1	0 0	0 0 1	1 1

b)



Rys. 13-26

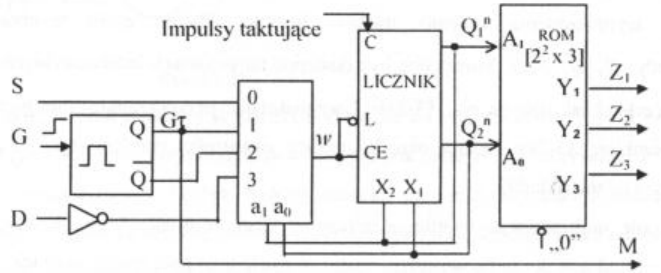
Szczególne właściwości programu rozważanego układu pozwalają na uproszczenie rozwiązania po zmianie matrycy połączeń na wejściach informacyjnych multipleksera:

1. Graf rozważanego układu (rys. 13-25a) posiada taką właściwość, że dla każdego stanu wejść inny jest warunek przejścia. Przy określaniu matrycy połączeń można zadbać o to, aby dany sygnał warunku *w* przyporządkowany został wejściu informacyjnemu multipleksera o adresie odpowiadającym kodowi stanu układu, w którym według grafu rozważany sygnał jest warunkiem przejścia. Wtedy pole multipleksera jest zbędne, bo odpowiadające mu sygnały są dostępne na wejściach adresowych pamięci (wyjściach licznika).
2. W każdym stanie układ dla wartości: $w = 0$ następuje przejście do aktualnego stanu układu. Zatem dla każdego stanu układu pole skoku odpowiada aktualnemu stanowi licznika i w mikroinstrukcji pole to jest zbędne. Odpowiadające mu sygnały są dostępne na wejściach adresowych pamięci.

Rozwiązanie uwzględniające modyfikacje opisane w punktach 1 i 2 przedstawia rys. 13-27.

Tablica zawartości pamięci

	Stan licznika A_1 A_0	Pole wyjść Y_1 Y_2 Y_3
①	0 0	1 0 0
②	0 1	0 1 0
③	1 0	0 0 0
④	1 1	0 0 1



Rys. 13-27

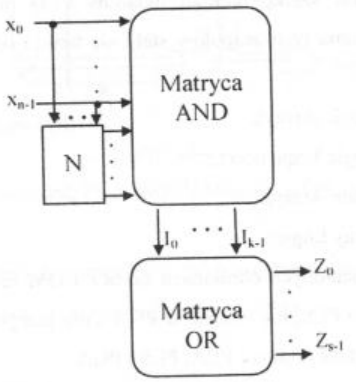
14. PROGRAMOWALNE MODUŁY LOGICZNE

Szybki rozwój technologii układów scalonych dużej skali integracji spowodował, że do realizacji układów cyfrowych coraz częściej stosuje się programowalne elementy logiczne posiadające liczne odmiany o wspólnej nazwie PLD (Programmable Logic Devices).

Omówiona wcześniej pamięć stała ROM (PROM, EPROM, E^2PROM) może być traktowana jako taki przedstawiciel układów PLD, w którym matryca iloczynów realizowana w postaci dekodera jest stała, realizuje wszystkie pełne iloczyny wejść adresowych jako adresy komórek pamięci. Umożliwia ona programowanie jedynie sum, a przez to realizację jedynie kanonicznej postaci funkcji logicznej.

Układ PLD dzięki możliwości programowania iloczynów i sum, umożliwia realizację minimalnej postaci funkcji logicznej. Zatem realizacja funkcji w oparciu o PLD może być bardziej oszczędna w porównaniu z realizacją w oparciu o pamięć ROM i daje możliwość eliminacji hazardu.

Produkowane w latach siedemdziesiątych układy PLA stanowiły strukturę o schemacie blokowym zgodnym z rys. 14-1, którą logicznie można ująć jako sieć bramek AND-OR (rys. 14-2). Programowanie ich odbywało się za pomocą odpowiedniej maski na etapie produkcji. Stanowiło to dużą niedogodność dla użytkownika i ograniczenie w stosowaniu układów PLA.

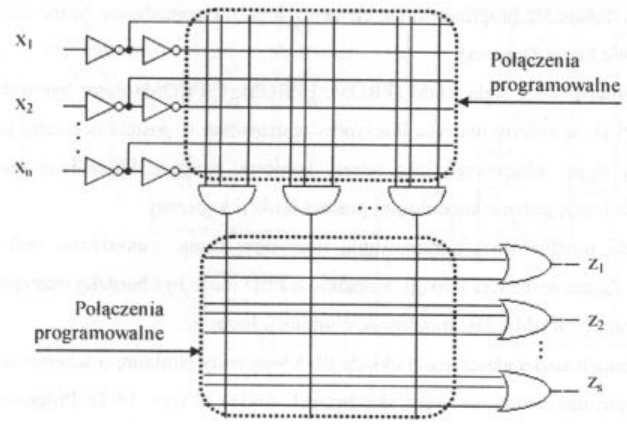


Rys. 14-1

W wyniku programowania układu o takiej strukturze: każde z wejść zewnętrznych (proste lub zanegowane) może być dołączone do dowolnej bramki AND, a każde z wyjść bramki AND może być dołączone do dowolnej bramki OR.

W układach PLA jednokrotnie programowalnych przez użytkownika na etapie produkcji realizowane są połączenia: każdej bramki AND z każdym wejściem zewnętrznym oraz każdej bramki OR z wyjściami każdej bramki AND.

Programowanie połączeń zależnie od technologii realizowane jest podobnie jak w pamięciach stałych (rozdział 12).



Rys. 14-2

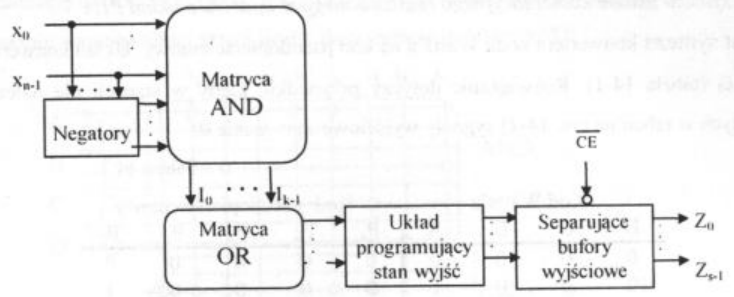
Obecnie produkowanych jest szereg odmian układów PLD programowanych przez użytkownika. Również urządzenia do programowania tych zespołów stały się tanie i dostępne. Do najbardziej popularnych należą:

1. PLA (ang. Programmable Logic Array),
2. PLS (ang. Programmable Logic Sequencer),
3. PGA (ang. Programmable Gate Array),
4. PAL (ang. Programming Array Logic).

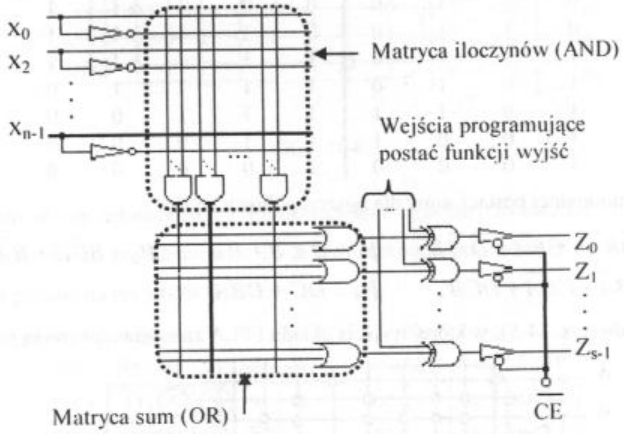
Matryca AND we wszystkich wymienionych odmianach oprócz ROM jest programowalna. Matryca OR jest programowalna w strukturach: PLA i PLS, nie istnieje w PGA i nie jest programowalna w PAL. Dalej zostaną omówione jedynie podstawowe struktury odmian: PLA, PLS i PGA.

14.1. Układy PLA

Układ o podstawowej strukturze PLA programowany przez użytkownika (Field PLA) składa się z programowalnej matrycy sum logicznych, programowalnej matrycy iloczynów logicznych oraz układów ustalających postać funkcji wyjść (rys. 14-3). Wewnętrzna struktura logiczna takiego układu przedstawiona jest na rys. 14-4.



Rys. 14-3



Rys. 14-4

Obecnie są dostępne układy programowalne PLA uwzględniające wiele udogodnień dla użytkownika. Między innymi umożliwiają one dodatkowo:

- programowanie części wyprowadzeń aby umożliwić zadeklarowanie ich jako wejścia lub wyjścia,
- programowanie połączeń wyjść układu z wejściami matrycy AND.

Synteza układu w oparciu o PLA sprowadza się do określenia minimalnych wyrażeń opisujących układ w postaci sumy lub postaci iloczynu. Dla realizacji funkcji F w postaci sumy matryca AND umożliwia realizację potrzebnych iloczynów jako składników sumy, a matryca OR pozwala na realizację sumy odpowiednich iloczynów potrzebnych do realizacji tej funkcji. Wyjścia układu należy wtedy zaprogramować jako proste. W przypadku, gdy funkcja przeciwna do F umożliwia jej prostszą realizację, można zrealizować funkcję $\overline{F}$ na wyjściach z matrycy OR a następnie zaprogramować wyjścia jako zanegowane.

***Przykład 14.1** – synteza układu kombinacyjnego realizowanego w oparciu o układ PLA*

Rozważana jest synteza konwertera kodu Watts'a na kod pseudopierścieniowy. Układ konwertera opisany jest tablicą zależności (tabela 14-1). Rozwiązanie dotyczy przypadku, kiedy w stanach nie należących do kodu Watts'a (nie ujętych w tabeli na rys. 14-1) sygnały wyjściowe są w stanie *0*.

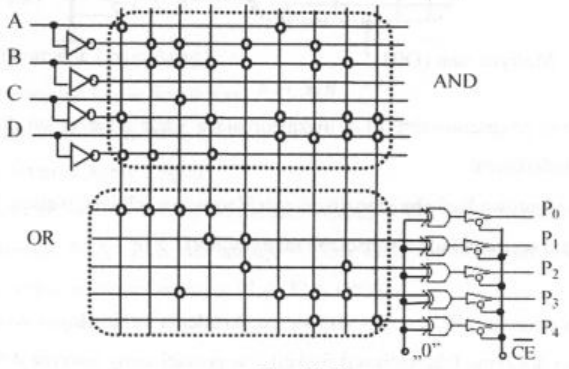
Tabela 14.1

Kod Watts'a				Kod pseudopierścieniowy				
D	C	B	A	P_4	P_3	P_2	P_1	P_0
0	0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	0	1
0	0	1	1	0	0	0	1	1
0	0	1	0	0	0	1	1	1
0	1	1	0	0	1	1	1	1
1	1	1	0	1	1	1	1	1
1	0	1	0	1	1	1	1	0
1	0	1	1	1	1	1	0	0
1	0	0	1	1	1	0	0	0
1	0	0	0	1	0	0	0	0

Z wolnej do hazardu minimalnej postaci sumy dla poszczególnych wyjść:

$$P_0 = \overline{D}\,\overline{C}A + \overline{D}B\overline{A} + CB\overline{A} + \overline{D}\,\overline{C}B, \quad P_1 = \overline{D}\,\overline{C}B + B\overline{A}, \quad P_2 = B\overline{C}D + B\overline{A},$$
$$P_3 = D\overline{C}A + DB\overline{A} + CB\overline{A} + D\overline{C}B, \quad P_4 = D\overline{C} + DB\overline{A}.$$

wynika realizacja układu (rys. 14-5), w której wyjścia układu FPLA zaprogramowane są jako proste.

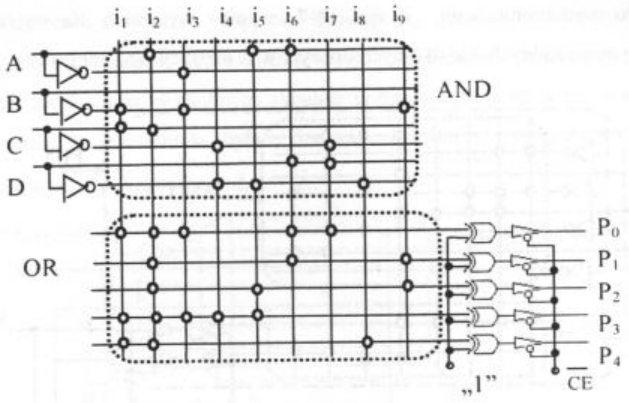


Rys. 14-5

W oparciu o wolną od hazardu minimalną postać iloczynu dla poszczególnych wyjść:

$$P_0 = (B + A)(\overline{D} + C)(\overline{C} + \overline{A})(\overline{C} + B)(\overline{D} + \overline{A}) = \overline{\overline{B}\,\overline{A} + D\overline{C} + CA + C\overline{B} + DA} = \overline{i3 + i7 + i2 + i1 + i6},$$
$$P_1 = B(\overline{D} + \overline{A})(\overline{C} + \overline{A}) = \overline{\overline{B} + DA + CA} = \overline{i9 + i6 + i2},$$
$$P_2 = B(D + \overline{A})(\overline{C} + \overline{A}) = \overline{\overline{B} + \overline{D}A + CA} = \overline{i9 + i5 + i2},$$
$$P_3 = (B + A)(D + C)(\overline{C} + \overline{A})(\overline{C} + B)(D + \overline{A}) = \overline{\overline{B}\,\overline{A} + \overline{D}\,\overline{C} + CA + C\overline{B} + \overline{D}A} = \overline{i3 + i4 + i2 + i1 + i5},$$
$$P_4 = D(\overline{C} + B)(\overline{C} + \overline{A}) = \overline{\overline{D} + C\overline{B} + C\overline{A}} = \overline{i8 + i1 + i2}.$$

można podać realizację układu (rys. 14-6), w której wyjścia układu PLA zaprogramowane są jako zanegowane. Ostatnie rozwiązanie wymaga takiej samej liczby iloczynów w matrycy AND.



Rys. 14-6

***Przykład 14.2** – synteza układu sekwencyjnego realizowanego w oparciu o układ PLA*

Działanie układu opisuje siatka programu (rys. 14-7a). Po zakodowaniu wierszy otrzymuje się siatkę stanów wewnętrznych układu podaną na rys 14-7b.

a)

y_1y_2 \ x_1x_2	00	01	11	10	W
00	(1)	(5)	(3)	2	0
01	4	5	(6)	(2)	0
11		-	(9)	2	1
10	(4)	5	9	(8)	1

b)

y_1y_2 \ x_1x_2	00	01	11	10	W
00	00	00	00	01	0
01	11	00	01	01	0
11	10	ΦΦ	11	01	1
10	10	00	11	10	1
			Y_1	Y_2	

c)

y_1y_2 \ x_1x_2	00	01	11	10
00	0	0	0	0
01	1	0	0	0
11	1	Φ	1	0
10	1	0	1	1
			Y_1	

d)

y_1y_2 \ x_1x_2	00	01	11	10
00	0	0	0	1
01	1	0	1	1
11	0	Φ	1	1
10	0	0	1	0
			Y_2	

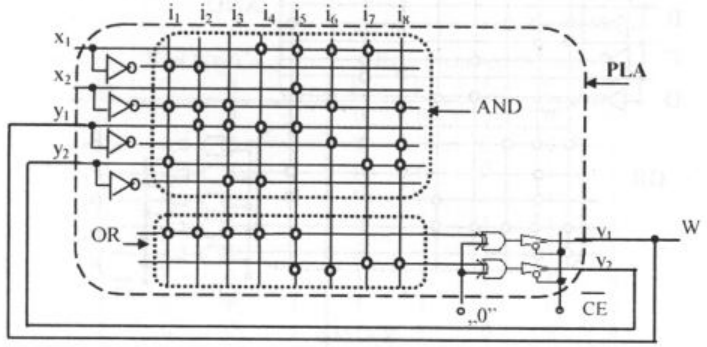
Rys. 14-7

Z grup zaznaczonych na rys. 14-7c i 14-7d wynikają minimalne wyrażenia opisujące Blok Pamięci:

$$Y_1 = \overline{y_2}\overline{x_1}\overline{x_2} + y_1\overline{x_1}\overline{x_2} + y_1\overline{y_2}\overline{x_2} + \overline{y_1}y_2x_1 + y_1x_1x_2 = i_1 + i_2 + i_3 + i_4 + i_5$$

$$Y_2 = \overline{y_1}x_1\overline{x_2} + y_1x_1x_2 + y_2\overline{x_1} + \overline{y_1}y_2\overline{x_2} = i_6 + i_5 + i_7 + i_8$$

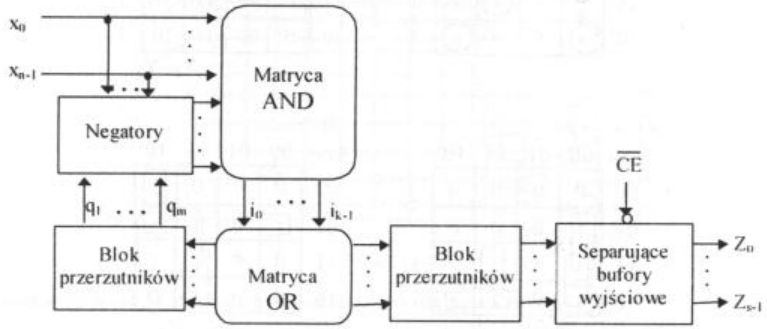
Dla realizacji BP potrzebne jest 8 iloczynów realizowanych w matrycy AND, dwóch sum w matrycy OR oraz zewnętrznych sprzężeń zwrotnych (rys. 14-8). Iloczyny i_2, i_3 realizują składniki antyhazardowe, niezbędne w bloku pamięci układu asynchronicznego. Z rys. 14-7c wynika wyrażenie dla wyjścia: $W = Y_1$. Zatem realizacja wyjścia nie wymaga żadnych dodatkowych iloczynów w matrycy AND ani sum w matrycy OR.



Rys. 14-8

14.2. Układy PLS

Na rys. 14-9 przedstawiono ogólną strukturę układu PLS. W stosunku do układu PLA układ PLS posiada dodatkowo przerzutniki synchroniczne lub asynchroniczne np. *sr*, jak na rys. 14-10. Połączenie matrycy AND z matrycą OR jest takie, jak w układzie PLA. Wyjścia matrycy OR sterują wejściami przerzutników *sr*. Część tych przerzutników stanowi wyjście układu, a wyjścia pozostałych przerzutników razem z wejściami zewnętrznymi doprowadzone są do matrycy AND w postaci prostej i zanegowanej.



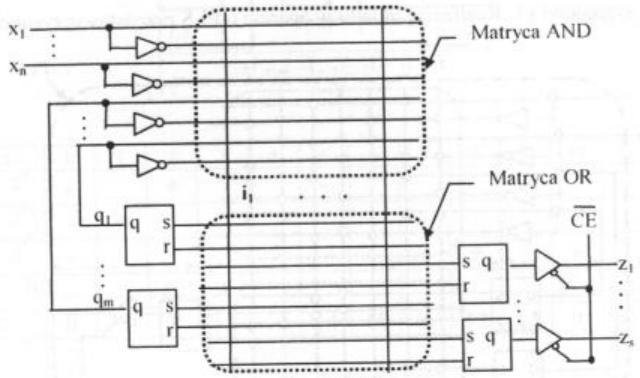
Rys. 14-9

Synteza układu realizowanego w oparciu o strukturę PLS polega na określeniu rozwiązania jedną z metod opisanych wcześniej w postaci wyrażeń o postaci alternatywnej (sumy) dla wejść przerzutników sr. Wyrażenia te nie konieczne muszą być minimalne, ponieważ:

- w matrycy AND koszt iloczynu mniej wejściowego nie jest mniejszy od kosztu iloczynu pełnego (opisanego wszystkimi sygnałami układu).

- stany niestabilne „1" i „0", w których są aktywne wejścia s i r, rzadko bywają sąsiednie logicznie, więc liczba składników postaci minimalnej (realizowanych jako iloczyny w matrycy AND) bywa zbliżona do liczby składników w postaci kanonicznej.

Jeżeli struktura nie posiada wyprowadzeń wyjść bezpośrednio z matrycy OR, to BKwy układu sekwencyjnego należy zrealizować w osobnym układzie w oparciu np. o PLA lub bramki. Realizacja funkcji kombinacyjnej w oparciu o przerzutniki byłaby nie ekonomiczna i wymagałaby dla każdego z wyjść Z określenia wejść przerzutnika w postaci: $s = Z$, $r = \overline{Z}$.



Rys. 14-10

Obecnie są dostępne układy programowalne z przerzutnikami uwzględniające wiele udogodnień dla użytkownika. Między innymi umożliwiają one dodatkowo:

- programowanie funkcji wejść przerzutnika: informacyjnych, ustawiających i zegarowego,
- programowanie typu przerzutnika,
- programowanie części wyprowadzeń aby umożliwić zadeklarowanie ich jako wejścia lub wyjścia,
- programowanie dodatkowej matrycy uzupełniającej.

***Przykład 14.3** – synteza układu sekwencyjnego w oparciu o układ PLS*

Rozważana jest synteza układu z przykładu 14.2. Etapy syntezy przedstawione na rys. 14-7a ÷14-7c są takie same jak w przypadku realizacji układu na zasadzie ogólnych sprzężeń zwrotnych w oparciu o PLA. Dalsze etapy dotyczące określenia wejść przerzutnika: s, r mogą przebiegać prościej, bez określania minimalnych wyrażeń.

Dla elementów pamięci Y_1 i Y_2 funkcje wejść s i r w kanonicznej postaci sumy, łatwo można określić bezpośrednio z siatek podanych na rys. 14-11, w oparciu o wyróżnione (przez pogrubienie) stany niestabilne tych elementów pamięci:

$$\text{dla } Y_1\text{:}\quad s = \overline{y_1}\,\overline{y_2}\,\overline{x_1}\,\overline{x_2} = i_1 \qquad r = y_1 y_2 x_1 \overline{x_2} + y_1 \overline{y_2}\,\overline{x_1}\,x_2 = i_2 + i_3$$

$$\text{dla } Y_2\text{:}\quad s = \overline{y_1}\,\overline{y_2} x_1 \overline{x_2} + y_1 \overline{y_2} x_1 x_2 = i_4 + i_5 \qquad r = \overline{y_1} y_2 \overline{x_1} x_2 + y_1 y_2 \overline{x_1}\,\overline{x_2} = i_6 + i_7$$

a)

y_1y_2 \ x_1x_2	00	01	11	10
00	0	0	0	0
01	**1**	0	0	0
11	1	Φ	1	**0**
10	1	**0**	1	1

Y_1

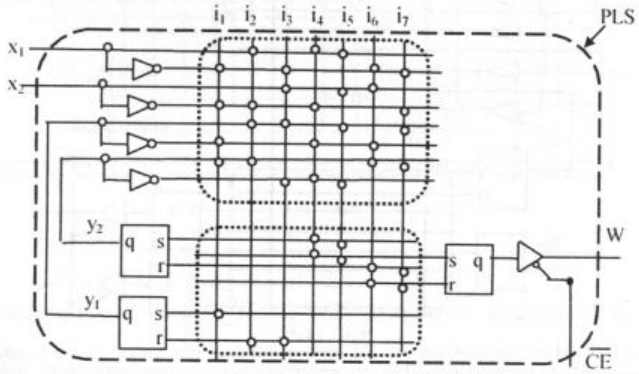
b)

y_1y_2 \ x_1x_2	00	01	11	10
00	0	0	0	**1**
01	1	**0**	1	1
11	**0**	Φ	1	1
10	0	0	**1**	0

Y_2

Rys. 14-11

Wyjście W odpowiada sygnałowi y1. Realizację układu w oparciu o PLS przedstawia rysunek 14-12.

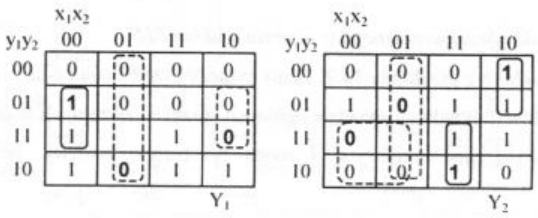


Rys. 14-12

Sprawdźmy, czy warto określić minimalną postać wyrażeń dla wejść przerzutników. Z grup zaznaczonych na rys. 14-13 minimalne wyrażenia:

$$dla\ Y_1: \quad s = y_2\overline{x_1}\,\overline{x_2}, \qquad r = \overline{x_1}x_2 + y_2x_1x_2,$$

$$dla\ Y_2: \quad s = \overline{y_1}x_1\overline{x_2} + y_1x_1x_2, \qquad r = \overline{x_1}x_2 + y_1\overline{x_1},$$

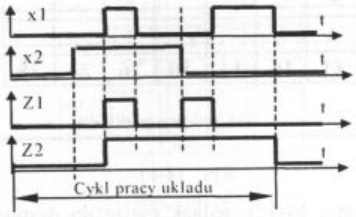


Rys. 14-13

W minimalnych postaciach sumy dla tych sygnałów występuje tylko o jeden iloczyn mniej niż w postaciach kanonicznych. Zatem dla realizacji układu w oparciu o układ PLS można pominąć często uciążliwy etap minimalizacji wyrażeń dla wejść przerzutników.

***Przykład 14.4** – synteza układu sekwencyjnego realizowanego w oparciu o PLS*

Działanie układu opisuje wykres czasowy na rys. 14-14 ten sam, co w przykładzie 9.5 na rys. 9-33, ale w programie jego pracy przyjęto inną kolejność zmian sygnałów Z1, Z2 niż w przykładzie 9.5 tak, jak to podaje tablica kolejności łączeń na rys. 14-15.



Rys. 14-14

Takty		0	1	2	3	4	5	6	7	8	9	10	11	00
x_1	2^0	-		+			-				+		-	
x_2	2^1	-	+						-					
z_1	2^2	-				+		-		+		-		
z_2	2^3				+									-
NSU		0	2	3	11	15	14	10	8_1	12	13	9	8_2	0-

Cykl pracy układu

Rys. 14-15

Tablica z rys. 14-15 zawiera sprzeczne powtórzenia stanów w cyklu pracy układu, więc nie jest rozwiązalną. W tablicy linią ciągłą zaznaczono takty należące do warunków działania i linią przerywaną takty należące do warunków niedziałania. Wskazują one, że występują sprzeczne powtórzenia stanu 8 w taktach: 7 i 11. Stany sprzeczne zaznaczono różniącymi się indeksami.

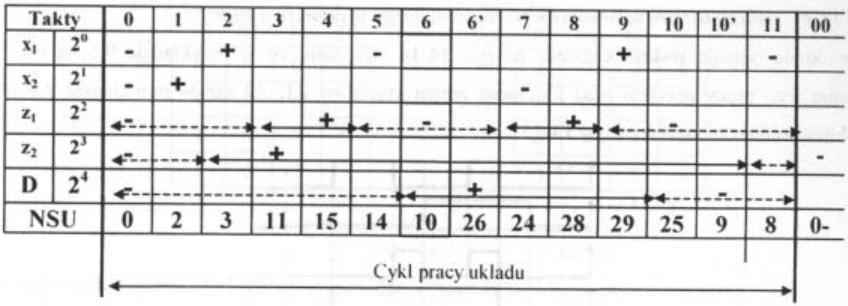
W kolejnej tablicy na rys. 14-16 wskazano podział cyklu pracy układu na części nie zawierające powtórzeń sprzecznych i uwzględniające warunki opisane w rozdziale dotyczącym tablic kolejności łączeń.

Takty		0	1	2	3	4	5	6	7	8	9	10	11	00
x_1	2^0	-		+			-				+		-	
x_2	2^1	-	+						-					
z_1	2^2	-				+		-		+		-		
z_2	2^3				+									-
NSU		0	2	3	11	15	14	10	8_1	12	13	9	8_2	0-

Cykl pracy układu

Rys. 14-16

Rozwiązalną TKŁ otrzymaną po wprowadzeniu elementu dodatkowego podaje rys. 14-17.

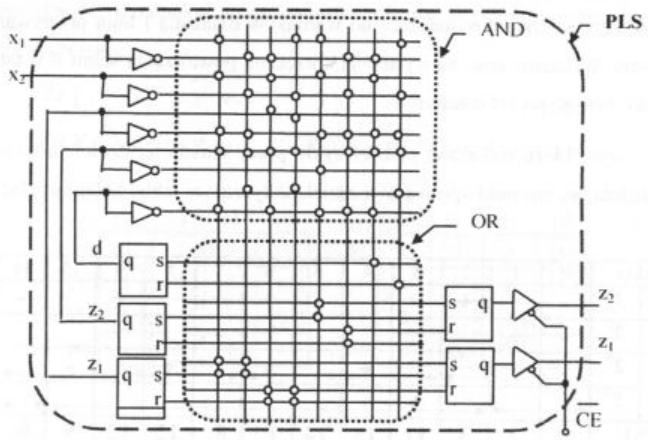


Rys. 14-17

Funkcja dla Z_1 jest kombinacyjna (rys. 14-19), jednak mając do dyspozycji strukturę z rysunku 14-18, tę funkcję też opłaca się zrealizować w oparciu o przerzutnik. Kanoniczną postaci sumy dla wejść przerzutników łatwo można określić bezpośrednio z rozwiązalnej TKŁ. Takty poprzedzające bezpośrednio zmianę na „+" rozważanego elementu należą do zbioru składników jedynki wejścia *s*. Takty poprzedzające bezpośrednio zmianę na „-" tego elementu należą do zbioru składników jedynki wejścia *r* . W rozważanym układzie:

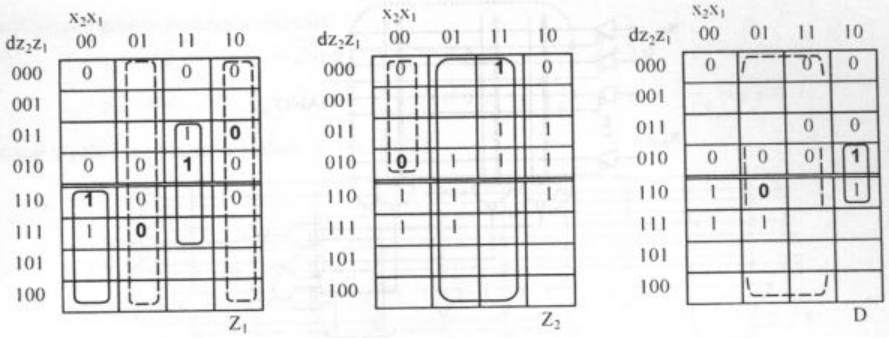
dla Z_1	dla Z_2	dla D
$s = \Sigma\,(11, 24)dz_2z_1x_2x_1$	$s = \Sigma\,(3)dz_2z_1x_2x_1$	$s = \Sigma\,(10)dz_2z_1x_2x_1$
$r = \Pi\,(14, 29)dz_2z_1x_2x_1$	$r = \Pi\,(8)dz_2z_1x_2x_1$	$r = \Pi\,(25)dz_2z_1x_2x_1$

Rozwiązanie układu z funkcjami wejść przerzutników w kanonicznej postaci podaje rys. 14-18.



Rys. 14-18

Sprawdźmy, czy rozwiązanie oparte o minimalne wyrażenia jest prostsze. Z warunków działania i warunków niedziałania rozwiązalnej TKŁ wynikają siatki dla elementów: Z_1, Z_2 i D podane na rys. 14-19.



Rys. 14-19

Z tych siatek wynikają minimalne wyrażenia dla funkcji wejść przerzutników *sr* :

dla Z_1: $s = z_2 x_2 x_1 + d\,\overline{x_2}\,\overline{x_1}$, $r = \overline{x_2}\,x_1 + x_2\overline{x_1}$,

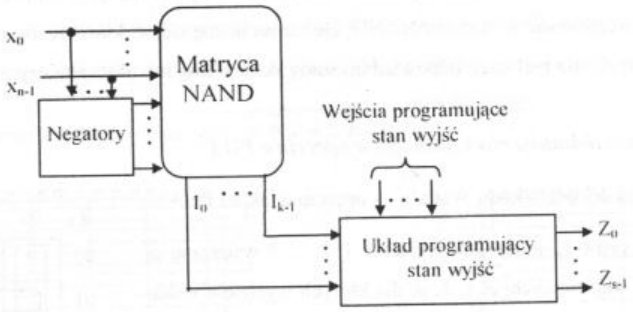
dla Z_2: $s = x_1$, $r = \overline{d}\,\overline{x_2}\,\overline{x_1}$,

dla D: $s = z_2\overline{z_1}\,x_2\overline{x_1}$ $r = x_1\overline{z_1}$.

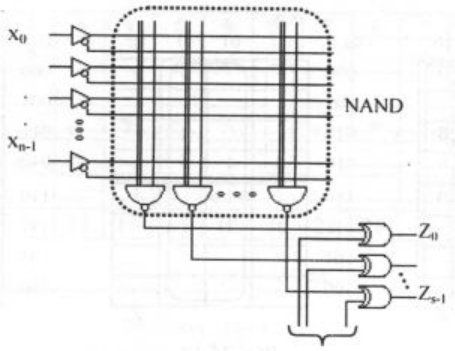
Wskazują one. że niezbędna liczba iloczynów w matrycy AND zmniejszy się tylko o jeden.

14.3. Układy PGA

Zespoły FPGA stanowią uproszczoną wersję zespołów PLA. W odróżnieniu od zespołu PLA. układ PGA posiada jedynie matrycę programowanych zanegowanych iloczynów logicznych (NAND) oraz układów ustalających postać funkcji wyjść. Schemat blokowy układu PGA przedstawia rys. 14-20, a schemat logiczny rys. 14-21. Układ PGA umożliwia realizację funkcji logicznych typu AND albo NAND.



Rys. 14-20



Wejścia programujące stan wyjść

Rys. 14-21

Algorytm syntezy układu kombinacyjnego przy użyciu układ PGA

1. W oparciu o opis słowny, wykres czasowy lub tablice zależności sygnałów wyjściowych od sygnałów wejściowych projektowanego układu należy określić kanoniczną postać funkcji wyjściowych.
2. Określić minimalną postać sumy lub iloczynu realizowanych funkcji wyjściowych i dokonać wyboru postaci dającej prostszą strukturę układu PGA.
3. W matrycy najpierw realizowane są poszczególne składniki dla minimalna postać sumy lub czynniki dla minimalna postać iloczynu, a następnie otrzymane sygnały wprowadzane są (zawracane są) na wejścia matrycy aby realizować funkcję wyjściową jako odpowiednio sumę lub iloczyn utworzonych wcześniej sygnałów.
4. Należy odpowiednio zaprogramować wyjścia układu w stan niski lub wysoki.

Liczba składników sumy lub odpowiednio czynników iloczynu ma wpływ nie tylko na liczbę zanegowanych iloczynów, które trzeba zrealizować w matrycy NAND, ale też na liczbę wejść, które trzeba przeznaczyć na zawrócenie odpowiednich wyjść dla realizacji odpowiednio sumy składników lub iloczynu czynników.

Przykład 14.5 *- synteza dekodera realizowanego w oparciu o PGA*

Rozważana jest synteza dekodera kodu Watts'a w oparciu o układ PGA.

W siatce na rys.14-22 indeksami wyjść: 0, 1, 2, ..., 9 wskazane są kombinacje sygnałów wejściowych: *d, c, b, a*, dla których wyjście o wskazanym indeksie ma wartość „1".

W stanach nie wskazanych przez żaden indeks wyjścia posiadają wartość dowolną (Φ).

d c \ b a	00	01	11	10
00	0	1	2	3
01				4
11				5
10	9	8	7	6

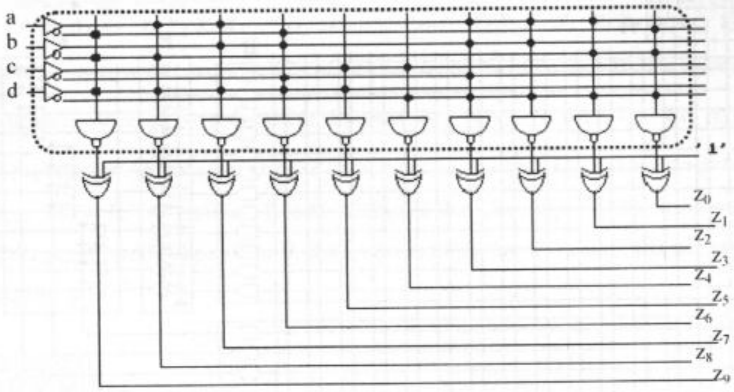
Rys. 14-22

Z zaznaczonych grup wynikają wyrażenia:

$$z_0 = \bar{d}\,\bar{b}\,\bar{a} \qquad z_1 = \bar{d}\,\bar{b}\,a \qquad z_2 = \bar{d}\,b\,a \qquad z_3 = \bar{d}\,\bar{c}\,b\,\bar{a} \qquad z_4 = \bar{d}\,c$$

$$z_5 = d\,c \qquad z_6 = d\,\bar{c}\,b\,\bar{a} \qquad z_7 = d\,b\,a \qquad z_8 = d\,\bar{b}\,a \qquad z_9 = d\,\bar{b}\,\bar{a}$$

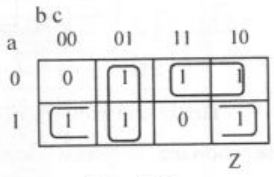
Z wyrażeń wynika rozwiązanie podane na rys. 14-23.



Rys. 14-23

Przykład 14.6

Rozważana jest realizacja układu opisanego siatką na rys 14-24.



Rys. 14-24

Minimalne wyrażenie alternatywne ma postać: $Z = a\bar{c} + \bar{a}b + \bar{b}c$,

inaczej:

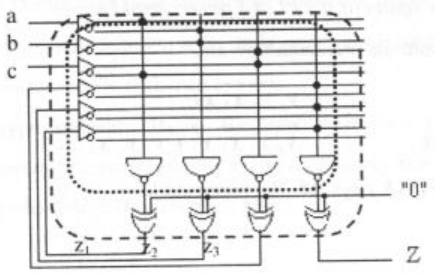
$$Z = a\bar{c} + \bar{a}b + \bar{b}c = z_1 + z_2 + z_3 = \overline{\overline{z_1 z_2 z_3}} = \overline{\overline{a\bar{c}} \cdot \overline{\bar{a}b} \cdot \overline{\bar{b}c}}.$$

Z niej wynika rozwiązanie podane na rys. 14-25a.

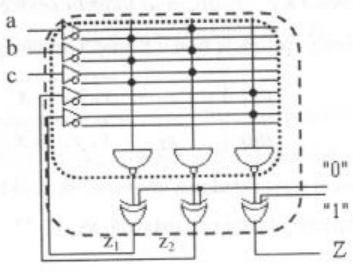
Minimalne wyrażenie koniunkcyjne ma postać: $Z = (a+b+c)\cdot(\bar{a}+\bar{b}+\bar{c})$. W oparciu o niego można podać prostsze rozwiązanie (rys. 14-25b) programując wyjście układu PLG dotyczące sygnału Z jako zanegowane.

Realizacja wtedy odpowiada wyrażeniu: $Z = \overline{\overline{a\,\bar{b}\,\bar{c} + a\,b\,c}} = \overline{\overline{a\,\bar{b}\,\bar{c}} \cdot \overline{a\,b\,c}}$

a)

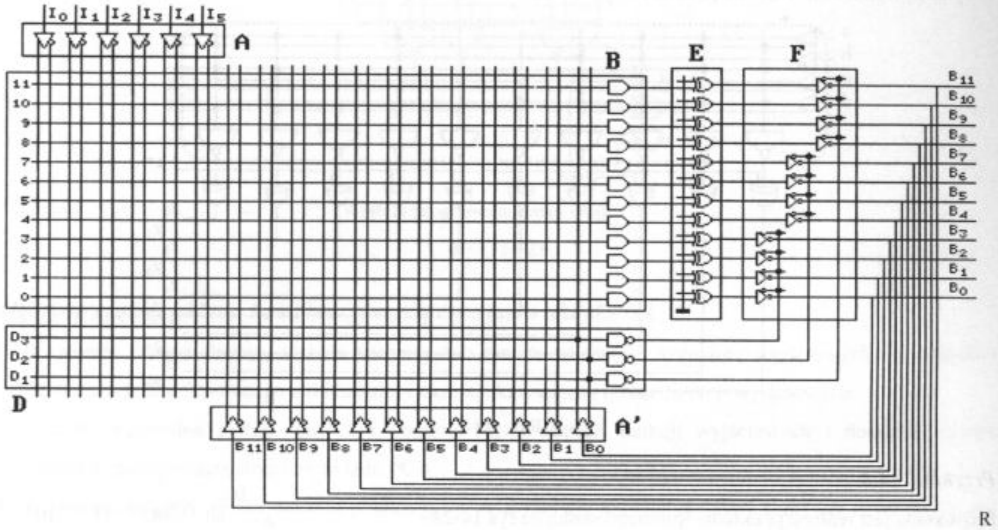


b)



Rys. 14-25

Obecnie produkowane są układy PGA o strukturze bardziej rozbudowanej niż opisana podstawowa struktura PGA. Można w niej programować nie tylko wejścia iloczynów i stan wyjść ale również: połączenie wyjść z wejściami matrycy NAND oraz deklarować część wyprowadzeń układu jako wejścia lub wyjścia doprowadzone do matrycy NAND. Przykładowe rozwiązanie przedstawia rysunek 14-26.



ys. 14-26

Dzięki dodatkowym układom struktura ma charakter uniwersalny. Posiada ona wejścia stałe (A) oraz programowane wejścia/wyjścia (A'). Każde z wejść jest doprowadzone do matrycy B w postaci prostej i zanegowanej. Wyjścia matrycy B można traktować jako wyjścia bramek AND. Matryca E pozwala zanegować sygnał wychodzący z matrycy B przez doprowadzenie sygnału odpowiadającego wartości logicznej „1" do odpowiednich wejść elementów Exclusive-OR. Bramki matrycy D steruję trójstanowymi buforami wyjściowymi w matrycy F. Zależnie od stanu sygnałów z matrycy D, wyprowadzenia: B_0, B_1, ..., B_{11} są wejściami układu lub negacją wyjść z matrycy E.

Przykład 14.7 – *realizacja układu sekwencyjnego w oparciu o FPGA i przerzutniki sr*

Rozważmy realizację bloku pamięci opisanego wyrażeniami dla przerzutników *sr*:

$$\text{dla } Y_1\text{:}\quad s_1 = \overline{y_1}\, y_2\, \overline{x_1}\, \overline{x_2} \qquad r_1 = y_1\, \overline{y_2}\, \overline{x_1}\, x_2$$

$$\text{dla } Y_2\text{:}\quad s_2 = \overline{y_1}\, \overline{y_2}\, x_1\, \overline{x_2} + y_1\, \overline{y_2}\, x_1\, x_2 \qquad r_2 = \overline{y_1}\, y_2\, \overline{x_1}\, x_2 + y_1\, y_2\, \overline{x_1}\, \overline{x_2}$$

w oparciu o przykładową uniwersalną strukturę układu FPGA z rys. 14-26 i przerzutniki *sr*.

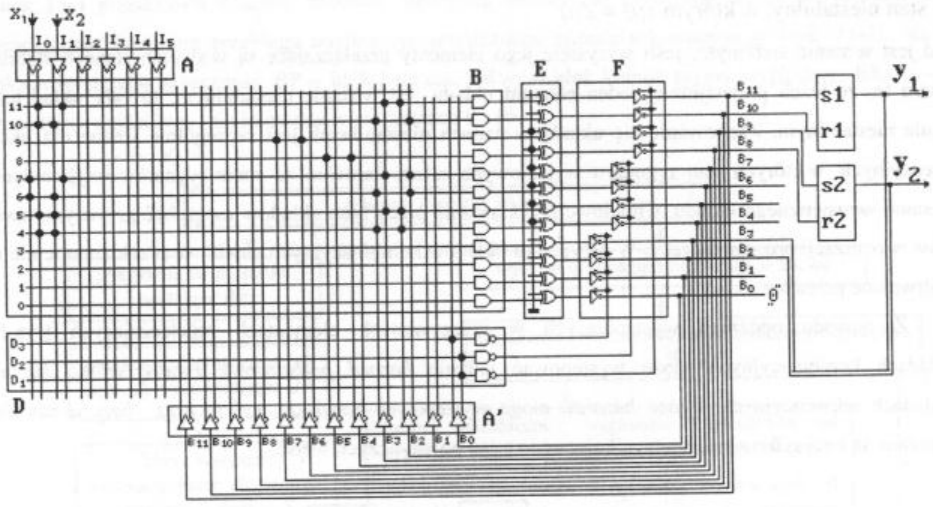
Rozwiązanie układu przedstawia rys. 14-27.

Iloczyny dla s_1 oraz r_1 są realizowane w matrycy B. Ponieważ w matrycy F odpowiadające im sygnały są negowane, dlatego w matrycy E przenoszone są jako zanegowane (wejścia bramek Exclusive-OR nie są połączone z sygnałem logicznego „0").

Suma iloczynów dla s_2 oraz dla r_2 jest realizowana w dwupoziomowym układzie NAND jako:

$$s_2 = \overline{\overline{y_1 y_2 x_1 x_2} \cdot \overline{y_1 y_2 x_1 x_2}} = \overline{i_1 \cdot i_2}, \qquad r_2 = \overline{\overline{y_1 y_2 x_1 x_2} \cdot \overline{y_1 y_2 x_1 x_2}} = \overline{i_3 \cdot i_4}.$$

Negacje iloczynów $i_1 \div i_4$ są realizowane na wyprowadzeniach odpowiednio $B_7 \div B_4$. Na wyprowadzeniu B_9 realizowana jest negacja iloczynu wyprowadzeń: B_7 oraz B_6 (czyli $\overline{i_1}$ oraz $\overline{i_2}$). Na wyprowadzeniu B_8 realizowana jest negacja iloczynu wyprowadzeń: B_5 oraz B_4 (czyli $\overline{i_3}$ oraz $\overline{i_4}$). Dla realizacji funkcji NAND na wyprowadzeniach: $B_7 \div B_4$ w matrycy E odpowiadające im elementy EXOR mają wejście połączone z sygnałem „0" dla przeniesienia odpowiednich iloczynów realizowanych w matrycy B bez negacji do matrycy F, w której są negowane.



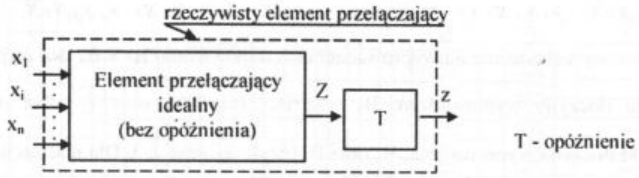
Rys. 14-27

Matryca D zapewnia takie sterowanie bramkami trójstanowymi w bloku F, aby:

- wyprowadzenia $B_{11} \div B_4$ były wyjściami (ponieważ $B_{11} \div B_8$ sterują przerzutnikami a $B_7 \div B_4$ są zawracane dla realizacji s_2 i r_2),
- wyprowadzenia $B_0 \div B_3$ były wejściami (ponieważ wyprowadzenia B_2, B_3, są potrzebne dla doprowadzenia sygnałów y_1 i y_2 do matrycy B a wyprowadzenie B_0, jest potrzebne dla doprowadzenia sygnału stałego 0 do matrycy D).

15. DYNAMIKA UKŁADÓW SEKWENCYJNYCH

W rzeczywistym elemencie przełączającym (rys. 15-1) funkcja logiczna $Z(x_1, ..., x_i, ..., x_n)$ realizowana jest z opóźnieniem τ : $z(t+\tau) = Z(t)$. Wartość z nazywamy stanem pracy układu, a wartość funkcji wynikającą ze stanów sygnałów wejściowych Z nazywany stanem wzbudzenia.



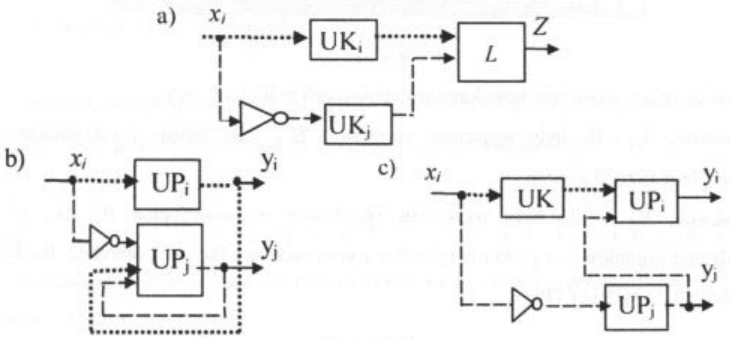
Rys. 15-1

W wyniku wystąpienia opóźnienia τ dla danego elementu możemy wyróżnić dwa stany:

- stan stabilny, w którym $z(t) = Z(t)$,
- stan niestabilny, w którym $z(t) \neq Z(t)$.

Układ jest w stanie stabilnym, jeśli wszystkie jego elementy przełączające są w swoich stanach stabilnych. Oznacza to, że jeżeli przynajmniej jeden element układu jest w stanie niestabilnym, to cały układ też jest w stanie niestabilnym. Zachowanie się układu w stanach niestabilnych jest szczególnie ważne dla układów sekwencyjnych, w których stan sygnałów wyjściowych zależy zarówno od stanu sygnałów wejściowych jak i od stanu wewnętrznego układu. Chwilowe przekłamania pracy tych układów mogą być przyczyną trwałych błędów w realizacji programu pracy. W przypadku układów kombinacyjnych chwilowe przekłamanie nie może być utrwalone przez ten układ.

Z powodu opóźnień występujących w rzeczywistych elementach przełączających (rys.15-1), w układach kombinacyjnych może występować jedynie *hazard podstawowy* zwany krótko *hazardem*. W układach sekwencyjnych oprócz *hazardu* mogą występować zjawiska: *wyścig* oraz *hazard zasadniczy*. Rozważane są tutaj tylko zjawiska wynikające ze zmian pojedynczych wejść.



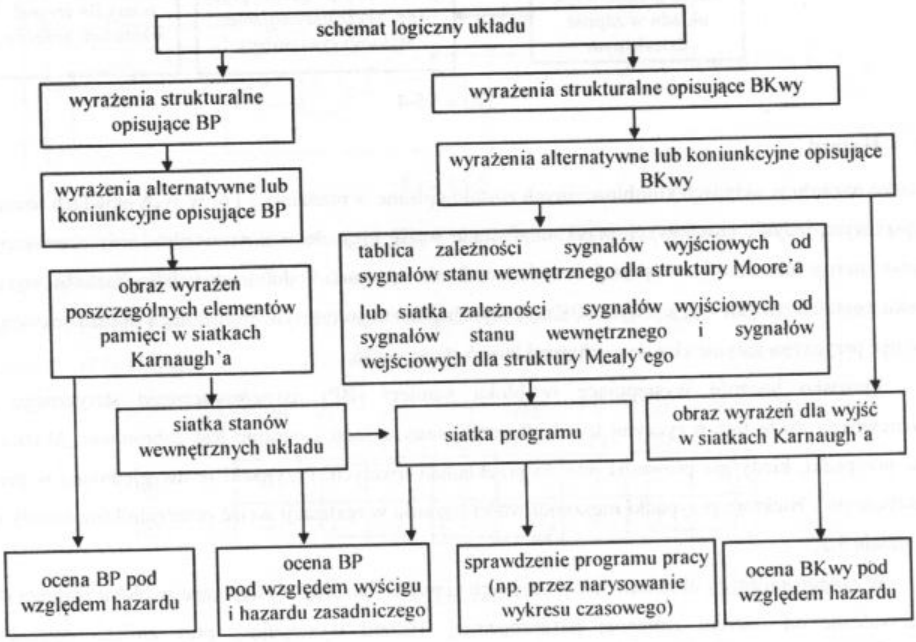
Rys. 15-2

Wspólną cechą tych trzech zjawisk jest to, że zmiany proste (na rys. 15-2 linie kropkowane) i zanegowane (na rys. 15-2 linie przerywane) pewnego sygnału x_i po drogach z różnym opóźnieniem sterują pewnym elementem przełączającym.

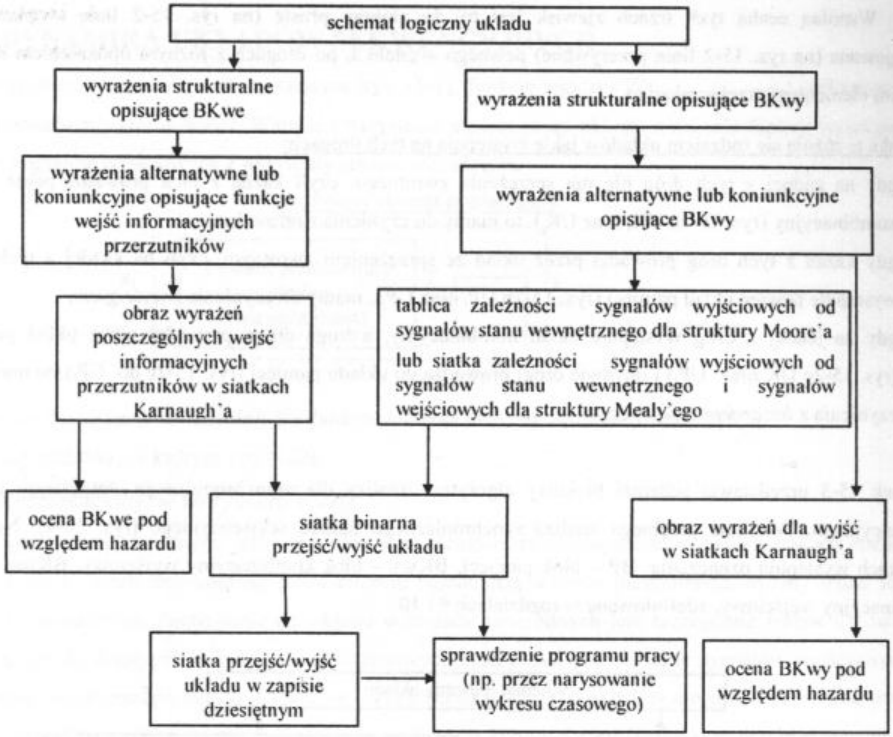
Zjawiska te różnią się rodzajem układów jakie występują na tych drogach:

- gdy na żadnej z tych dróg nie ma sprzężenia zwrotnego, czyli każda z nich prowadzi przez układ kombinacyjny (rys. 15-2a UK_i oraz UK_j), to mamy do czynienia z *hazardem*,
- gdy każda z tych dróg prowadzi przez układ ze sprzężeniem zwrotnym, czyli na każdej z tych dróg występuje pewien układ pamięci (rys. 15-2b UP_i oraz UP_j), mamy do czynienia z *wyścigiem*,
- gdy na jednej z dróg występuje układ kombinacyjny, a druga droga prowadzi przez układ pamięci (rys. 15-2c UK oraz UP_j) i obydwie drogi prowadzą do układu pamięci (rys. 15-2c do UP_i), to mamy do czynienia z *hazardem zasadniczym*.

Rysunek 15-3 przedstawia schemat blokowy algorytmu analizy dla asynchronicznego statycznego układu sekwencyjnego. Podobnie przebiega analiza synchronicznego układu sekwencyjnego (rys. 15-4). Na tych rysunkach występują oznaczenia: BP – blok pamięci, BKwy – blok kombinacyjny wyjściowy, BKwe – blok kombinacyjny wejściowy, zdefiniowane w rozdziałach 9 i 10.



Rys. 15-3



Rys. 15-4

15.1 Hazard

Zjawisko hazardu w układach kombinacyjnych zostało opisane w rozdziale 3 i 6. W tych układach hazard może być przyczyną jedynie chwilowych przekłamań stanu wyjść i jest dozwolone wtedy, kiedy rozważany układ cyfrowy steruje obiektem o większej bezwładności niż on sam. Podobnie zjawisko hazardu występujące w bloku kombinacyjnych wyjściowym (BKwy) asynchronicznego lub synchronicznego układu sekwencyjnego może być przyczyną jedynie chwilowych przekłamań stanu wyjść.

Zjawisko hazardu występujące w bloku pamięci (BP) *asynchronicznego* statycznego układu sekwencyjnego może być przyczyną trwałych przekłamań układu i ogólnie jest zabronione. Można jednak podać przypadki, kiedy nie prowadzi ono do przekłamań trwałych. Przypadki te uwzględnione w programie dydaktycznym[1]. Niektóre przypadki nieszkodliwości hazardu w realizacji wejść przerzutników zostały opisane w rozdziale 9.5.

W *synchronicznych* układach sekwencyjnych zjawisko hazardu występujące w bloku pamięci (BP) ma skutki zależne od rodzaju zmiennej hazardogennej. Hazard występujący przy zmianie sygnału stanu wewnętrznego układu nie może powodować trwałych przekłamań pracy układu. Wynika to z następujących

[1] program dydaktyczny Qn-h-szk.zip dostępny na witrynie internetowej http://zmitac.iinf.polsl.gliwice.pl/

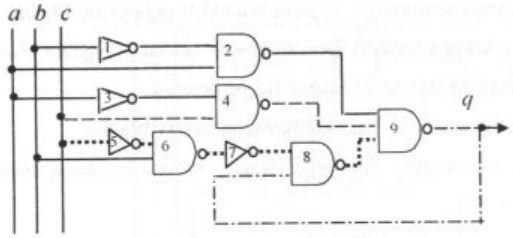
rozważań. Podstawowym warunkiem poprawnej pracy układu jest, aby nadążał on za zamianami wejść. Zatem układ synchroniczny musi nadążać za zmianami sygnału zegarowego. Jeżeli zmiana stanu układu występująca przy pewnym impulsie zegarowych powoduje chwilowe przekłamanie w wyniku hazardu, to ten stan nieustalony musi się zakończyć przed kolejnym impulsem zegarowym i nie może być przyczyną przejścia do błędnego stanu układu.

Skutki chwilowego przekłamania spowodowane hazardem występującym w BKwe układu synchronicznego przy zmianie wejścia układu (nie będącego sygnałem zegarowym) zależą od czasu trwania tego przekłamania i parametrów czasowych przerzutników zastosowanych w układzie. Dla niekorzystnych proporcji opóźnień trwałe przekłamania w tym przypadku nie są wykluczone.

Dla zabezpieczenia układu przed hazardem, w etapie minimalizacji normalnych wyrażeń logicznych opisujących układ tworzone są wyrażenia antyhazardowe. Eliminacja przekłamań spowodowanych hazardem jest również możliwa przez wprowadzenie opóźnień odpowiednich zboczy sygnałów.

Przykład 15.1

Na rys. 15-5 przedstawiony jest schemat logiczny badanego układu oraz opisujące go wyrażenia: strukturalne i o postaci alternatywnej (otrzymane po zastosowaniu prawa negacji).



$$Q = \overline{\overline{a \cdot \bar{b}} \cdot \overline{\bar{a} \cdot c} \cdot \overline{\bar{c} \cdot b} \cdot q}$$

$$Q = a \cdot \bar{b} + \bar{a} \cdot c + \bar{c} \cdot b \cdot q$$

Rys. 15-5

Rys. 15-6 przedstawia obraz wyrażenia dla Q w siatce Karnaugha. Wynika z niego, że badany układ posiada HSd przy przejściach między stanami wyrażonymi:

binarnie: $(1011 \leftrightarrow 1010)_{qabc}$, $(1110 \leftrightarrow 1100)_{qabc}$, $(1001 \leftrightarrow 1101)_{qabc}$, $(0001 \leftrightarrow 0101)_{qabc}$

dziesiętnie: $(11 \leftrightarrow 10)_{qabc}$, $(14 \leftrightarrow 12)_{qabc}$, $(9 \leftrightarrow 13)_{qabc}$, $(1 \leftrightarrow 5)_{qabc}$

	a b c	(0)	(1)	(3)	(2)	(6)	(7)	(5)	(4)
	q	000	001	011	010	110	111	101	100
(0)	0	0	1	1	0	0	0	1	1
(8)	1	0	1	1	1	1	0	1	1

Q

Rys. 15-6

Hazard przy przejściu: $(1 \leftrightarrow 5)_{qabc}$ nie powinien nas interesować, ponieważ to przejście nie jest możliwe. Stany: $(1)_{qabc}$ i $(5)_{qabc}$ są stanami niestabilnymi, a jednym z warunków poprawnej pracy układu jest aby układ nadążał za zmianami wejść (patrz podstawowe warunki poprawnej pracy układu w rozdziale 9.1). Z każdego z tych

stanów układ powinien przejść najpierw do odpowiadającego mu stanu stabilnego: $(9$ lub $13)_{qabc}$ zanim wystąpi jakakolwiek zmiana wejść.

Hazard przy przejściu: $(9 \leftrightarrow 13)_{qabc}$, może powodować jedynie chwilowe przekłamanie w postaci przejścia do stanu $(1)_{qabc}$ lub $(5)_{qabc}$. Z tych stanów niestabilnych układ powróci do odpowiednio: $(9)_{qabc}$ lub $(13)_{qabc}$.

Hazard przy przejściach: $(11 \leftrightarrow 10)_{qabc}$ i $(14 \leftrightarrow 12)_{qabc}$ może powodować trwałe przekłamanie w postaci przejścia do stanu błędnego odpowiednio: $(2)_{qabc}$ lub $(6)_{qabc}$. Te błędne stany mogą być utrwalone.

Przykładowo prześledźmy dokładniej przejście $(11 \leftrightarrow 10)_{qabc}$. Przejście to następuje po zmianie sygnałów wejściowych: $(011 \rightarrow 010)_{abc}$ dla $q = 1$. Przed rozważaną zmianą układ znajduje się w stanie stabilnym wyrażonym dwójkowo: $(1011)_{qabc}$, czyli dziesiętnie $(11)_{qabc}$. Zmiana sygnału $c = 1 \rightarrow 0$, może powodować chwilową błędną wartość $q = 0$ w wyniku hazardu przy rozważanym przejściu. Ta błędna wartość może być następnie utrwalona w pętli sprzężenia zwrotnego. Ilustrują to linie przerywana, kropkowana i kropka-kreska na rys. 15-5. Przyjmijmy równe opóźnienie bramek składowych układu.

Opóźnienie zmian sygnału c na drodze wiodącej przez bramkę z numerem 4 (linia przerywana) do bramki z numerem 9 jest mniejsze niż opóźnienie tego sygnału do bramki 9 na drodze wiodącej przez bramki z numerami: 5, 6, 7 i 8. (linia kropkowana). Z tego powodu na wyjściu bramki 9 pojawi się błędny stan „0". Ten błędny stan „dociera" do bramki 8 (linia kropka-kreska) wcześniej niż zmiany sygnału c na drodze wiodącej przez bramki: 5, 6 i 7 (linia kropkowana). Można to uzasadnić liczbą bramek:

- od wejścia c do bramki 8 na drodze zaznaczonej linią kropkowaną są trzy bramki.
- od wejścia c do bramki 8 na drodze zaznaczonej linią przerywaną i kropka-kreska są dwie bramki.

Zatem błędny stan będzie utrwalony.

Jeżeli dla projektowanego układu nie przeprowadza się szczegółowych badań skutków hazardu tak jak np. w przykładzie 15.1. to dla każdego elementu pamięci należy wyeliminować hazard przy każdym przejściu.

15.2 Wyścig

Zjawisko wyścigu w układach *synchronicznych* może być przyczyną jedynie chwilowych przekłamań stanu wyjść. Stan nieustalony jaki występuje przy zmianie stanu układu po pewnym impulsie zegarowych musi się zakończyć przed kolejnym impulsem zegarowym, więc nie może mieć wpływu na następny stan układu.

Zjawisko wyścigu zostało uwzględnione w omówieniu *asynchronicznych* statycznych układów sekwencyjnych w rozdziale 9.3. W tych układach wyścig może być przyczyną trwałych przekłamań. Mamy z nim do czynienia wtedy, gdy w układzie istnieje bezpośrednie przejście między stanami niesąsiednimi logicznie. Rozróżniane są dwa rodzaje wyścigu:

niekrytyczny - gdy wszystkie zmiany układu możliwe z powodu wyścigu prowadzą do zgodnego z programem stanu stabilnego,

krytyczny - gdy istnieje co najmniej jedna z dróg prowadząca do niezgodnego z programem stanu stabilnego.

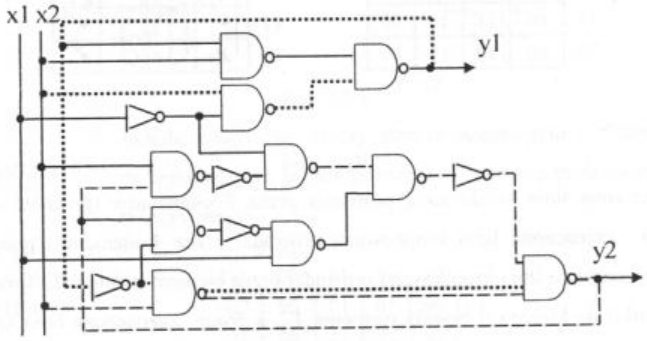
Oceniając skutki wyścigu krytycznego w oparciu o siatkę stanów wewnętrznych i siatkę programu można sądzić, że układ po „trafieniu" w błędny stan stabilny, powinien utrwalić ten stan. Badając dokładnie skutki tego zjawiska, można wykazać, że po przejściu układu do błędnego stanu stabilnego w wyniku wyścigu krytycznego może on się zachować inaczej. Mianowicie może on oscylować między stanem błędnym i poprawnym. Jest to zilustrowane w przykładzie 15.2.

Projektant układów cyfrowych nie musi tak szczegółowo badać skutków wyścigu. Dla niego nie jest ważne, czy w wyniku wyścigu krytycznego układ przechodząc do błędnego stanu stabilnego pozostanie w nim, czy też będzie oscylował między stanem błędnym i poprawnym. Jedno i drugie zachowanie się układu jest wtedy błędne i wyścig krytyczny należy eliminować.

Jednak badając układ zawierający wyścig warto wiedzieć, że może się on zachować inaczej niż to wynika z siatki przejść i siatki programu.

Złym skutkom wyścigu można zapobiegać przez odpowiednie zakodowanie stanów wewnętrznych układu, pokierowanie przejściami dla uniknięcia wyścigu krytycznego dobierając odpowiednie wartości funkcji logicznych w stanach niestabilnych lub przez odpowiedni dobór opóźnień.

Przykład 15. 2



Rys. 15-7

Przeanalizujmy działanie układu[2] z rys. 15-7. Układowi temu odpowiadają wyrażenia strukturalne:

$$Y1 = \overline{\overline{x1 \cdot x2} \cdot \overline{x2 \cdot y1}} \qquad Y2 = \overline{\overline{x1 \cdot x2 \cdot y2} \cdot \overline{x1 \cdot y2 \cdot y1} \cdot \overline{y1 \cdot x2}}$$

Korzystając z prawa negacji otrzymujemy wyrażenia:

$$Y1 = \overline{x1} \cdot x2 + x2 \cdot y1, \qquad Y2 = \overline{x1} \cdot x2 \cdot y2 + x1 \cdot y2 \cdot \overline{y1} + \overline{y1} \cdot x2$$

W oparciu te wyrażenia można wypełnić siatki stanów wewnętrznych poszczególnych elementów pamięci, a następnie siatkę stanów wewnętrznych całego układu (rys. 15-8a). Rys. 15-8b przedstawia siatkę przejść układu. W siatce tej: kółkiem zaznaczony jest każdy stan stabilny, kropką zaznaczony jest każdy stan

[2] Dokładną analizą tego układu umożliwia program dydaktyczny sym_haz.zip dostępny na witrynie internetowej http://zmitac.iinf.polsl.gliwice.pl/

niestabilny, a strzałką z linią ciągłą zaznaczone jest każde przejście ze stanu niestabilnego do odpowiadającego mu stanu stabilnego. Ponadto strzałkami z linią przerywaną zaznaczone są przejścia wynikające z wyścigu.

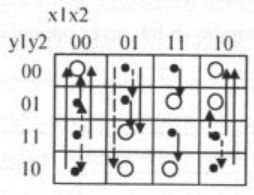
Z tych siatek (rys. 15-8) wynika, że w analizowanym układzie występuje wyścig niekrytyczny w stanie niestabilnym $(1100)_{y_1y_2x_1x_2}$ oraz wyścigi krytyczne w stanach niestabilnych: (0001 i $1110)_{y_1y_2x_1x_2}$.

Dokładna analiza stanów układu w siatce na rys. 15-8 pozwala zauważyć, że wyścig krytyczny w stanie $(1110)_{y_1y_2x_1x_2}$ nie powinien nas niepokoić. Układ w żaden sposób nie może trafić do tego stanu. Nie ma bowiem stanu stabilnego, sąsiedniego logicznie ze względu na zmianę wejść do stanu rozważanego. A rozważane są tutaj tylko zjawiska wynikającej z pojedynczych zmian wejść.

Zbadajmy zachowanie się układu po przejściu do stanu $(0001)_{y_1y_2x_1x_2}$, w którym występuje wyścig krytyczny, dla przypadku równych opóźnień bramek. Do rozważanego stanu układ może trafić ze stanu stabilnego $(0000)_{y_1y_2x_1x_2}$ po zmianie sygnału $x_2 = 0 \rightarrow 1$. Na rys 15-7 zaznaczono drogi, z różnicami opóźnień będące przyczyną błędnej pracy układu przy tym przejściu.

y1y2 \ x1x2	00	01	11	10
00	00	11	01	00
01	00	11	01	01
11	00	11	10	00
10	00	10	10	00
			Y_1	Y_2

b)



Rys. 15-8

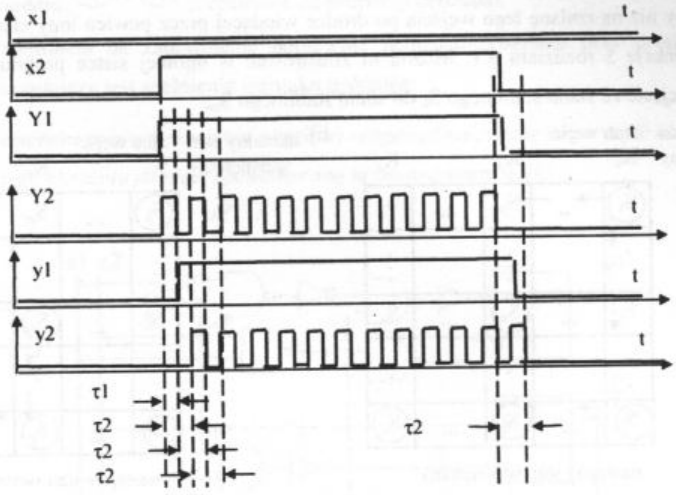
Droga zaznaczona linia kreskowaną prowadzi przez 7 elementów (przenosi zmiany zanegowane sygnału x_2), a droga zaznaczona linią kropkowaną prowadzi przez 4 elementy (przenosi zmiany proste sygnału x_2). Droga zaznaczona linią kropkowaną prowadzi przez element pamięci *Y1* (przez 2 bramki) i dalej (przez kolejne 2 bramki) do końcowej bramki elementu *Y2*, a droga zaznaczona linią kreskowaną prowadzi przez element pamięci *Y2* (przez 2 bramki) i dalej w sprzężeniu zwrotnym do końcowej bramki elementu *Y2* (przez kolejnych 5 bramek). Ponieważ droga zaznaczona linią kropkowaną jest krótsza w czasie niż droga zaznaczona linią kreskowaną, to układ po przejściu do stanu niestabilnego $(0001)_{y_1y_2x_1x_2}$ następnie przejdzie do stanu błędnego $(1001)_{y_1y_2x_1x_2}$. W tym stanie wymuszana jest jako następna wartość $Y2 = 0$, a układ miał przejść do stanu o wartości $Y2 = 1$. Zatem przy rozważanym przejściu w realizacji programu może być trwałe przekłamanie.

Z siatki stanów wewnętrznych i siatki programu wynika, że układ po trafieniu w błędny stan stabilny $(1001)_{y_1y_2x_1x_2}$ w wyniku wyścigu krytycznego, powinien utrwalić ten stan.

Dokładne badanie układu przy tym przejściu pozwala stwierdzić, że po przejściu do błędnego stanu $(1001)_{y_1y_2x_1x_2}$, stan ten nie koniecznie musi być utrwalony. Dla sygnału *Y2* mogą występować oscylacje. Układ może oscylować między stanem błędnym $(1001)_{y_1y_2x_1x_2}$ i poprawnym $(1101)_{y_1y_2x_1x_2}$. Wyjaśnia to wykres czasowy na rys. 15-9. Na tym rysunku uwzględniono:

τ1 - opóźnienie z jakim element pamięci o wyjściu *y1* reaguje na zmianę wejścia *x2*, czyli opóźnienie zmiany stanu pracy w stosunku do zmiany stanu wzbudzenia dla *y1*,

τ2 - opóźnienie zmiany stanu pracy *y2* w stosunku do zmiany stanu jego wzbudzenia *Y2*, przy założeniu że opóźnienie dla narastającego zbocza jest zawsze takie samo i równe opóźnieniu dla opadającego zbocza.



Rys. 15-9

Takie zachowanie się układu (oscylacje między stanem poprawnym i błędnym po przejściu do błędnego stanu z powodu wyścigu krytycznego), można potwierdzić zarówno podczas badań tego układu na symulatorze[3], jak i badań układu rzeczywistego.

y1y2 \ x1x2	00	01	11	10
00	00	01	01	00
01	00	11	01	01
11	00	11	10	10
10	00	10	10	00
			Y1	Y2

Rys. 15-10

Dla uniknięcia złych skutków wyścigu krytycznego, należy pokierować przejściem wybierając jeden ze sposobów:

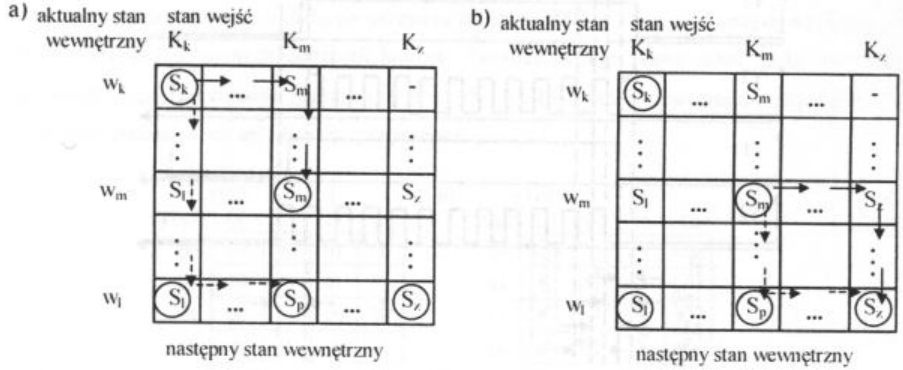
- zmieniając funkcję logiczną dla *Y1* na zgodną z siatką na rys. 15-10,
- zwiększając opóźnienie na drodze wiodącej przez *Y1* np. przez dodanie parzystej liczby bramek o wymaganym opóźnieniu.

[3] symulację umożliwia program dydaktyczny zawarty w Haz_zas.zip dostępny na witrynie internetowej http://zmitac.iinf.polsl.gliwice.pl/

15.3 Hazard zasadniczy

Hazard zasadniczy może występować w układach posiadających dwa lub więcej elementów pamięci. Przez element pamięci należy rozumieć fragment BP realizujący pewien sygnał stanu wewnętrznego.

Aby program układu określony siatką programu był poprawnie realizowany, każdy z elementów pamięci powinien z mniejszym opóźnieniem reagować na zmianę wejścia x_i po drodze wiodącej do niego przez układ kombinacyjny niż na zmianę tego wejścia po drodze wiodącej przez pewien inny element pamięci. Jest o tym mowa w punkcie 3 rozdziału 9.1. Można to zilustrować w ogólnej siatce programu na rys. 15-11. Rozważane jest przejście ze stanu stabilnego S_k do stanu stabilnego S_m.



Rys. 15-11

Aby to przejście było zrealizowane poprawnie (na rys. 15-11a droga zaznaczona linią ciągłą wiodąca przez stan niestabilny S_m), wszystkie elementy pamięci układu powinny z mniejszym opóźnieniem zauważyć zmianę stanu wejść (kolumny): $K_k \rightarrow K_m$ niż zmianę pewnego elementu pamięci, wyrażonego w siatce jako zmiana stanu (wiersza) $W_k \rightarrow W_l$.

W przeciwnym przypadku, gdy pewien element pamięci, który przy tym przejściu nie powinien zmieniać swego stanu, wcześniej zauważy zmianę stanu układu (wiersza): $W_k \rightarrow W_m$ niż zmianę stanu wejść (kolumny) $K_k \rightarrow K_m$, następuje przejście ze stanu stabilnego S_k w wierszu W_k i kolumnie K_k do stanu niestabilnego S_l w wierszu W_l i kolumnie K_k. Może to spowodować przejście do stanu stabilnego S_l w wierszu W_l. Z opóźnieniem zauważona zmiana stanu wejść (zmiana kolumny) na K_m powoduje wtedy błędne przejście do końcowego stanu S_p zamiast do S_m. Droga odpowiadająca błędnej pracy układu w siatce na rys. 15-11a zaznaczona jest strzałkami z linią przerywaną.

Obydwie opisane drogi: poprawna i błędna nie spotkają się w tym samym stanie stabilnym. Świadczy to, że przy rozważanym przejściu mamy do czynienia z hazardem zasadniczym.

Jeżeli obydwie opisane drogi: poprawna (zaznaczona linią ciągłą) i błędna (zaznaczona linią przerywaną) spotkają się w tym samym stanie stabilnym, to w wyniku błędnej proporcji opóźnień może wystąpić jedynie chwilowe przekłamanie. Taki przypadek dotyczy przejścia np. między stanami stabilnymi S_m i S_z. Na rys. 15-11b liniami: ciągłą przez stan niestabilny S_z i przerywaną przez stan stabilny S_p zilustrowano

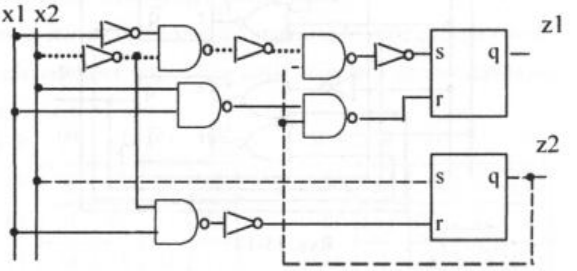
drogę odpowiednio poprawną i błędną przy tym przejściu. Obydwie drogi prowadzą do stanu stabilnego S_z. Zatem przy rozważanym przejściu nie ma hazardu zasadniczego.

Podobnie przy przejściu między stanami stabilnymi S_m i S_l nie ma hazardu zasadniczego. Oznacza to, że przejścia: $S_m \rightarrow S_z$ oraz $S_m \rightarrow S_l$ będą prowadziły ostatecznie do poprawnego stanu końcowego odpowiednio: S_l i S_z niezależnie od proporcji opóźnień.

Jedynym sposobem na zapewnienie poprawnej realizacji programu przy przejściu, przy którym występuje hazard zasadniczy jest spełnienie warunku opóźnień:

Każdy z elementów pamięci powinien wcześniej reagować na zmianę stanu wejść niż na zmianę innego elementu pamięci spowodowaną tą zmianą stanu wejść.

Przykład 15. 3



Rys. 15-12

Układ przedstawiony na rys.15-12 badany jest pod względem hazardu zasadniczego. Opisują go wyrażenia:

$$s1 = \overline{\overline{x1 \cdot x2 \cdot z2}} = \overline{x1} \cdot \overline{x2} \cdot z2 \qquad s2 = x2$$

$$r1 = \overline{\overline{x1 \cdot x2} \cdot z2} = x1 \cdot x2 + \overline{z2} \qquad r2 = \overline{\overline{x2} \cdot x1} = \overline{x2} \cdot x1$$

Na ich podstawie można określić siatkę stanów wewnętrznych układu oraz siatkę programu (rys. 15-13).

a)

z1z2 \ x1x2	00	01	11	10
00	00	01	01	00
01	11	01	01	00
11	11	11	01	10
10	00	01	01	00
			Z1	Z2

b)

z1z2 \ x1x2	00	01	11	10
00	(1)	2	5	(6)
01	3	(2)	(5)	6
11	(3)	(4)	5	6
10	1	2	5	6

Rys. 15-13

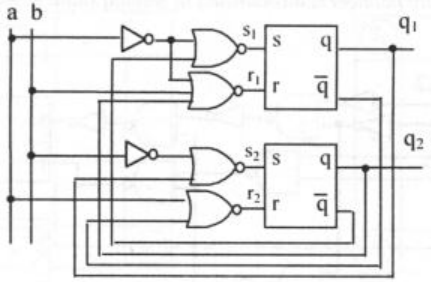
Wyniki analizy przejścia ze stanu stabilnego 1 do stanu stabilnego 2 wskazują, że przy tym przejściu występuje hazard zasadniczy. Poprawna droga, zgodna z programem, prowadzi (linia ciągła) ze stanu stabilnego 1 wzdłuż wiersza do stanu niestabilnego 2, a następnie wzdłuż kolumny do stanu stabilnego 2. Droga błędna prowadzić może (linia przerywana) ze stanu stabilnego 1 wzdłuż kolumny, przez stan niestabilny 3, do stanu stabilnego 3, a następnie wzdłuż wiersza do stanu stabilnego 4.

Na schemacie logicznym (rys. 15-12) zaznaczono drogi sygnału *x2* wiodące do *Z1* przez układ kombinacyjny (linia kropkowana) i przez element pamięci *Z2* (linia przerywana).

Dla równych opóźnień bramek, opóźnienie na drodze zaznaczonej linią kropkowaną jest większe niż opóźnienie na drodze zaznaczonej linią przerywaną. Zatem na wejściu *s* przerzutnika *Z1* pojawi się na chwilę stan logiczny „1" i przerzutnik o wyjściu *Z1* utrwali sta: $Z1 = 1$. Układ przejdzie więc do błędnego stanu: $(11)_{Z1Z2}$ zamiast do stanu: $(01)_{Z1Z2}$. Takie zachowanie się układu można potwierdzić zarówno podczas badań tego układu na symulatorze[4], jak i badań układu rzeczywistego.

Przykład 15. 4

Dla układu o podanym schemacie (rys. 15-14) przeprowadzić analizę pod względem hazardu zasadniczego.



Rys. 15-14

Ze schematu wynikają wyrażenia opisujące wejścia przerzutników:

$$s_1 = q_2 \cdot a \qquad s_2 = \overline{q}_1 \cdot b$$

$$r_1 = \overline{q}_2 \cdot a \cdot \overline{b} \qquad r_2 = q_1 \cdot \overline{a}$$

Na podstawie tych wyrażeń można wypełnić siatki stanów wewnętrznych przerzutników. W stanach, w których $s = 1$, stan wzbudzenia elementu pamięci Q równa się 1, a w stanach, w których $r = 1$, stan wzbudzenia Q tego elementu równa się 0. Rys. 15-15 przedstawia siatki wypełnione tylko w aktywnych stanach wejść *s* lub *r*.

a)

q_1q_2 \ a b	00	01	11	10
00				0
01			1	1
11			1	1
10				0

Q_1

b)

q_1q_2 \ a b	00	01	11	10
00		1	1	
01		1	1	
11	0	0		
10	0	0		

Q_2

Rys. 15-15

W pozostałych stanach, w których $s = r = 0$, stan wzbudzenia Q elementu pamięci równy jest jego stanowi pracy. Rys. 15-16 przedstawia siatki całkowicie wypełnione dla obu przerzutników. W oparciu o nie można utworzyć siatkę stanów wewnętrznych układu (rys. 15-17a).

[4] symulację umożliwia program dydaktyczny zawarty w Haz_zas.zip dostępny na witrynie internetowej http://zmitac.iinf.polsl.gliwice.pl/

a)

q_1q_2 \ a b	00	01	11	10
00	0	0	0	0
01	0	0	1	1
11	1	1	1	1
10	1	1	1	0

Q_1

b)

q_1q_2 \ a b	00	01	11	10
00	0	1	1	0
01	1	1	1	1
11	0	0	1	1
10	0	0	0	0

Q_2

Rys. 15-16

Korzystając z siatki stanów wewnętrznych układu można utworzyć siatkę programu (rys. 15-17b). Tworzy się ją w następujący sposób:

- pola siatki, które mają takie same stany Q_1Q_2 co stany $q1q2$ w tym wierszu, tworzą stany stabilne układu (kolejnym stanom stabilnym odpowiadają kolejne numery),
- pola siatki, które mają stany Q_1Q_2 różne od stanów $q1q2$ dla danego wiersza tworzą stany niestabilne (numeracja stanów niestabilnych odpowiada ściśle numeracji stanów stabilnych.

a)

q_1q_2 \ a b	00	01	11	10
00	00	01	01	00
01	01	01	11	11
11	10	10	11	11
10	10	10	10	00

b)

q_1q_2 \ a b	00	01	11	10
00	(1)	4	5	(2)
01	(3)	(4)	5'	6
11	7	8	(5)	(6)
10	(7)	(8)	(9)	2

Rys. 15-17

Analizując siatkę programu można podać przejścia, w których występuje zjawisko hazardu zasadniczego. W siatce na rysunku 15-17b zilustrowano drogi: błędną (linią przerywaną) i poprawną (linią ciągłą) dla przykładowego przejścia zawierającego hazard zasadniczy: ze stanu stabilnego 4 do stanu stabilnego 5.

Jeżeli układ będąc w stanie stabilnym 4 zareaguje wcześniej na zmianę stanu sygnału wejściowego a, niż na zmianę stanu q_1 spowodowaną zmianą a, wtedy ze stanu stabilnego 4 poprzez stan niestabilny 5' przejdzie on do poprawnego stanu stabilnego 5. Taka praca jest poprawna. Kiedy układ najpierw zareaguje na zmianę stanu q_1 spowodowaną zmianą sygnału a (przejście ze stanu stabilnego 4 poprzez stan niestabilny 8 do stanu stabilnego 8) a później zareaguje na zmianę stanu wejściowego a (przejście ze stanu stabilnego 8 do stanu stabilnego 9), wtedy znajdzie się on w końcowym stanie stabilnym niezgodnym z programem.

W badanym układzie oprócz omawianego przejścia ze stanu stabilnego 4 do stanu stabilnego 5, są jeszcze inne przejścia zawierające hazard zasadniczy:

- ze stanu stabilnego 3 do stanu stabilnego 6, zagrożone przejściem do błędnego stanu stabilnego 2,
- ze stanu stabilnego 6 do stanu stabilnego 7, zagrożone przejściem do błędnego stanu stabilnego 1,
- ze stanu stabilnego 9 do stanu stabilnego 2, zagrożone przejściem do błędnego stanu stabilnego 6.

Wykaz literatury

[1] Kamionka-Mikuła H.: *Algorytm wyznaczania miejsc zmian elementów dodatkowych w tablicy kolejności łączeń*, AITiS. T-8/ Z.3-4, PAN, Warszawa 1997.

[2] Kamionka-Mikuła H.: *Optymalny sposób wprowadzania dodatkowych elementów pośredniczących do nierozwiązalnej tablicy kolejności łączeń*. AAiT , T-XXII/ Z-4, Warszawa 1977.

[3] Kamionka-Mikuła H.: *Analiza trzy-poziomowych układów NAND i NOR z uwzględnieniem zagadnień hazardu*. Praca oddana do druku w AITiS. PAN, Warszawa 1999.

[4] Łuba T., Jasiński K.: Zbierzchowski B.: *Specjalizowane układy cyfrowe w strukturach PLD i FPGA*. WKŁ, Warszawa 1997.

[5] Majewski W., Łuba T., Jasiński K., Zbierzchowski B.: *Programowalne moduły logiczne w syntezie układów cyfrowych*. WKŁ, Warszawa 1992.

[6] Praca zbiorowa pod redakcją H. Małysiaka: *Teoria automatów cyfrowych – Laboratorium*. WPŚl., Gliwice 1997.

[7] Praca zbiorowa pod redakcją H. Małysiaka i B. Pochopienia: *Układy cyfrowe – Zadania*. WPŚl., Gliwice 1995.

[8] Praca zbiorowa pod redakcją H. Małysiaka i J. Siwińskiego: *Zbiór zadań z układów przełączających*. WPŚl., Gliwice 1997.

[9] Praca zbiorowa pod kierunkiem H. Małysiaka: *Układy przełączające w automatyce przemysłowej – Zadania*. WNT, Warszawa 1981.

[10] Molski M.: *Modułowe i mikroprogramowalne układy cyfrowe*. WkiŁ, Warszawa 1986.

[11] Pieńkos J., Turczyński J.: *Układy scalone TTL w systemach cyfrowych*. WkiŁ, Warszawa 1986.

[12] Pochopień B.: *Arytmetyka systemów cyfrowych*. WPŚl., Gliwice 2000.

[13] Sandige R. S.: *Modern digital design*, McGraw-Hill Publishing Company, New York 1990.

[14] Siwiński J.: *Układy przełączające w automatyce*, WNT Warszawa 1980.

[15] Traczyk W.: *Układy cyfrowe. Podstawy teoretyczne i metody syntezy*. WNT, Warszawa 1986.

[16] Programy dydaktyczne wykonane w ramach prac dyplomowych pod kierunkiem Haliny Kamionki-Mikuły dostępne na witrynie internetowej: http://zmitac.iinf.polsl.gliwice.pl

Anal-wiz - zrealizowany przez Katarzynę Lecybyl i Anetę Porębską (Zander), przeznaczony dla komputerowej analizy wyrażenia logicznego (normalnego i różnego od normalnego) pod względem: poprawności dla zadanej funkcji, skracalności i hazardu. Uwzględnia aspekt dydaktyczny poprzez wizualizację obrazu wyrażenia w siatce Karnaugh'a.

Fpga.zip - zrealizowany przez Karinę Kanię, dotyczy nauczania realizacji układów w oparciu o programowane matryce logiczne typu FPGA.

Fpls.zip - zrealizowany przez Annę Tomaszewską, dotyczy realizacji układów w oparciu o programowane matryce logiczne typu FPLS.

Haz-stat.zip - zrealizowany przez Frih'a Naceur'a, dotycz nauczania zagadnień hazardu statycznego.

H-st-win.zip - zrealizowany przez Piotra Sitko, dotycz nauczania zagadnień hazardu statycznego.

Haz-sym.zip - zrealizowany przez Tomasza Żaka, uwzględnia symulator wybranych układów cyfrowych, celem nauczania zagadnień: hazardu, wyścigu, hazardu zasadniczego.

Haz-zas.zip - zrealizowany przez Artura Moskwę, dotyczy nauczania zagadnień hazardu zasadniczego.

Huf-dyd.zip - zrealizowany przez Krystynę Piontek, dotycz ilustracji syntezy układów asynchronicznych sekwencyjnych metodą Huffmana.

Kaz-wiz.zip - zrealizowany przez Grzegorza Nowaka, przeznaczony do komputerowej minimalizacji wyrażeń logicznych w oparciu o metodę Kazakowa z uwzględnieniem hazardu – zawiera wizualizację kolejnych etapów minimalizacji.

Mikr-nau.zip - zrealizowany przez Przemysława Filipczuka, dotyczy nauczania zagadnień układów mikroprogramowanych.

Mikr-syn.zip - zrealizowany przez Karolinę Student, przeznaczony do komputerowej syntezy układów mikroprogramowanych z wizualizacją kolejnych etapów syntezy.

Min-siat.zip - zrealizowany przez Agnieszkę Jelonek (Kozak), przeznaczony do nauczania inimalizacji wyrażeń logicznych w oparciu o metodę siatek Karnaugh'a, uwzględnia kompurerową minimalizację wyrażeń logicznych.

Min-tn.zip - zrealizowany przez Annę Filipowicz, przeznaczony do komputerowej minimalizacji wyrażeń logicznych w oparciu o metodę tablic niezgodności z uwzględnieniem hazardu.

Mult-syn.zip - zrealizowany przez Anetę Dąbek (Warmuz), dotyczy komputerowej syntezy układów realizowanych w oparciu o multipleksery i demultipleksery, uwzględnia komputerową minimalizację wyrażeń logicznych w oparciu o metodę bezpośredniego przeszukiwania i wizualizację jej kolejnych etapów.