

POMOC DYDAKTYCZNA DO WYKŁADU
TEORIA UKŁADÓW LOGICZNYCH

**PRZYGOTOWAŁ DR ANDRZEJ KALIŚ
WROCŁAW, SEMESTR ZIMOWY 2002/2003**

1 WPROWADZENIE

1.1 Wstęp

Układy elektroniczne dzielimy na układy analogowe i dyskretne. Ich podstawową rolą jest przetwarzanie sygnałów wejściowych w sygnały wyjściowe w ściśle określony sposób. Sygnały analogowe mają charakter ciągły, czyli ich amplituda może przybierać dowolną wartość, a zmiana jednej wartości w drugą następuje w sposób ciągły. Sygnały dyskretne nie są ciągłe. Zazwyczaj w ustalonym czasie, sygnał dyskretny przyjmuje ustaloną wartość. W niniejszym podręczniku będziemy się zajmować sygnałami cyfrowymi dwuwartościowymi, które są sygnałami dyskretnymi. Sygnał cyfrowy dwuwartościowy to taki, w którym można wyróżnić dwie wartości amplitudy, umownie nazywane: „jedyneką”, której może odpowiadać poziom napięcia np. $+3.3[V]$, oraz „zerem”, której reprezentacją jest poziom napięcia, np. $0[V]$. Układy dyskretne przetwarzające sygnały cyfrowe nazywamy układami logicznymi (cyfrowymi, przełączającymi).

Podstawową zaletą układów cyfrowych w stosunku do układów analogowych jest stosunkowo duża odporność na zakłócenia, wynikająca z samej natury przetwarzania sygnałów, jak i możliwości odpowiedniego ich projektowania. Z tego powodu układy cyfrowe stosuje się w układach elektronicznych najwyższej jakości, nawet tam, gdzie zastosowanie układów analogowych mogłoby się wydawać bardziej uzasadnione (np. telekomunikacja, teletransmisja danych, miernictwo).

Modelami matematycznymi układów przełączających są automaty skończone (*finite automata*). Model ten opiera się na skończonym zbiorze elementów, które podlegają ściśle określonym przekształceniom. Jeżeli elementy te tworzą pewne obiekty abstrakcyjne, to taki kierunek badań automatów skończonych nazywamy abstrakcyjną teorią automatów skończonych[1]. Jeśli abstrakcyjnej teorii automatów nadamy pewną treść logiczną, taką, by przy wykorzystaniu podstawowych zależności funkcjonalnych można było opisać sposób i metodę przetwarzania sygnałów dyskretnych zgodnie z założonym algorytmem, to taki kierunek badań nazywamy strukturalną teorią automatów[2].

Wykorzystywanym w podręczniku aparatem formalnym (w większości materiału) jest algebra Boole’a (wyrażenie boolowskie, funkcja boolowska). Układy logiczne można opisywać za pomocą formuł algebry Boole’a oraz po nałożeniu pewnych ograniczeń realizacyjnych umożliwiając określenie fizycznej struktury układu składającego się z elementów logicznych pochodzących z pewnej bazy elementowej o ściśle określonych własnościach.

Głównym celem podręcznika jest przedstawienie wybranych modeli matematycznych, w dosyć naturalny sposób dających się transponować w schemat logiczny układu. Unika się zbytniej formalizacji zagadnień. Podręcznik jest wstępem do studiowania układów cyfrowych średniego stopnia scalenia oraz architektur komputerów.

1.2 Podstawowe pojęcia języka matematycznego.

Autor podręcznika zakłada, że Czytelnikowi znane są podstawowe pojęcia języka matematycznego. Jednak, by ustrzec się od dwuznacznego rozumienia stosowanych w podręczniku pojęć, krótko je przypomnimy (Czytelnik, który opanował treść podręczników [3] może ten rozdział pominąć).

1. Każda teoria matematyczna jest zbiorem powiązanych ze sobą zdań, między którymi zachodzą pewne związki. Rachunek zdań jest teorią matematyczną badającą zdania wchodzące w skład teorii matematycznych oraz związki między nimi. Zdanie logiczne jest to

takie zdanie oznajmujące, o którym jednoznacznie możemy powiedzieć, że jest prawdziwe lub fałszywe. Zdaniu prawdziwemu będziemy przypisywać wartość prawda, a fałszywemu wartość fałsz.

2. Główną rolę w teoriach matematycznych odgrywają zdania złożone ze zdań prostych za pomocą spójników, takich jak lub, i, jeżeli ... to, wtedy i tylko wtedy, gdy, nie. Łącząc dwa zdania za pomocą jednego z pierwszych czterech spójników otrzymujemy nowe zdanie poprawne. Zdanie złożone otrzymane przez połączenie dwóch lub więcej zdań spójnikiem „lub” nazywamy alternatywą albo sumą logiczną zdań składowych. Podobnie, łącząc zdania proste spójnikiem „i”, otrzymujemy zdanie złożone nazywane koniunkcją albo iloczynem logicznym zdań składowych. Implikacją zdań składowych nazywamy zdanie złożone powstałe przez połączenie zdań prostych za pomocą spójnika „jeżeli...to”. Jest to bodaj najczęściej spotykana w matematyce postać zdania. Zdanie uzyskane przez połączenie dwóch zdań zwrotem „wtedy i tylko wtedy, gdy” jest nazywane równoważnością. Spójnik „nie” napisany przed jakimkolwiek zdaniem tworzy wraz z nim nowe zdanie, zwane zaprzeczeniem (negacją).
3. Spójniki wykorzystywane w rachunku zdań mają następujące oznaczenia symboliczne:

I	Lub	Wtedy i tylko wtedy, gdy	Jeżeli ... to	nie
\wedge	\vee	\Leftrightarrow	\Rightarrow	$-$

W literaturze spotykane są również inne sposoby oznaczania spójników logicznych, jednak w niniejszym podręczniku wykorzystujemy powyżej podane.

4. Wyrażenia mające budowę zdań i zawierające zmienne, za które można podstawić nazwy obiektów z ustalonego zbioru, są nazywane funkcjami zdaniowymi albo predykatami. Jeżeli w funkcji zdaniowej na miejsce zmiennej podstawimy odpowiednie nazwy, to otrzymamy zdanie prawdziwe lub fałszywe. Funkcje zdaniowe możemy ze sobą łączyć spójnikami logicznymi.
5. Kwantyfikatorami nazywamy zwroty:

- a) istnieje takie x, że ...,
- b) dla każdego x.

Łączenie kwantyfikatorów ze spójnikami, stałymi i zmiennymi umożliwia wypowiedzenie myśli matematycznej. Pierwszy z kwantyfikatorów nazywany kwantyfikatorem ogólnym i oznaczanym przez $(\forall x)$. Drugi kwantyfikator nazywany natomiast kwantyfikatorem szczegółowym i oznaczamy go przez $(\exists x)$. Jeżeli chcemy powiedzieć - istnieje jedyne x, takie że, to będziemy pisać $(\exists! x)$. Przykładowe zdanie postaci: dla każdego x, y istnieje jedyny z taki, że $x+z=y$ i $y+z=x$ możemy zapisać w następujący sposób:

$$(\forall x, y)(\exists! z)(x + z = y) \wedge (y + z = x) .$$

6. Parą uporządkowaną dwóch elementów (x, y) nazywamy parę, w której gra rolę kolejność występowania elementów. Element x jest pierwszym elementem, a element y drugim. Zbiór F, którego elementami są wszystkie uporządkowane pary (x, y) , gdzie $x \in X$, a $y \in Y$, nazywamy produktem kartezjańskim i oznaczamy $X \times Y$, czyli

$$X \times Y = \{(x, y) \mid x \in X \wedge y \in Y\} .$$

7. Podzbiór produktu kartezjańskiego $X \times Y$ nazywać będziemy relacją określoną na zbiorze X. Fakt, że para $(x, y) \in X \times Y$ zapisujemy xRy . Zbiór X nazywamy dziedziną, zbiór Y zaś przeciwdziedziną relacji R.
8. Relacja równości = zachodzi w zbiorze X jedynie między dowolnym przedmiotem a nim samym, czyli jest to relacja określona przez zbiór postaci (x, x) .

9. Relację R określoną na zbiorze X nazywamy równoważnością, jeżeli spełnia ona warunki:

- $(\forall x \in X)(xRx)$ zwrotność,
 $(\forall x, y \in X)(xRy \Rightarrow yRx)$ symetria,
 $(\forall x, y, z \in X)(xRy \wedge yRz \Rightarrow xRz)$ przechodniość.

10. Relację R określoną na zbiorze X nazywamy relacją porządku jeżeli spełnia ona warunki:

- $(\forall x \in X)(xRx)$ zwrotność,
 $(\forall x, y \in X)(xRy \wedge yRx \Rightarrow x = y)$ antysymetria,
 $(\forall x, y, z \in X)(xRy \wedge yRz \Rightarrow xRz)$ przechodniość.

11. Relację R porządku w zbiorze X nazywamy relacją liniowego porządku, jeżeli spełnia następującą własność:

- $(\forall x, y \in X)(xRy \vee yRx)$ spójność.

12. Funkcją f nazwiemy pewien podzbiór produktu $X \times Y$ taki, że

- $(\forall x \in X)(\exists y \in Y)((x, y) \in f)$
 $((x, y_1) \in f \wedge (x, y_2) \in f) \Rightarrow y_1 = y_2$

13. Niech zbiór X będzie zbiorem n -elementowym, zbiór Y zaś zbiorem m -elementowym. Liczba funkcji określonych na zbiorze X , których wartości należą do zbioru Y wynosi m^n .

14. Działaniem n -argumentowym określonym na zbiorze X nazywamy funkcję $f(x_1, \dots, x_n)$, która każdemu zestawowi elementów x_1, \dots, x_n (w skrócie n -ka) ze zbioru X przyporządkowuje pewien element y ze zbioru X . Element y nazywamy wartością tej funkcji dla argumentów x_1, \dots, x_n i oznaczamy przez $f(x_1, \dots, x_n)$. Działaniem zeroargumentowym nazywamy ustalony element zbioru. Działania można ze sobą składać i otrzymywać nowe działania.

15. Termem nazwiemy wyrażenie zbudowane w specjalny sposób ze zmiennych i symboli oznaczających działania. Innymi słowy, jest to napis wyznaczający schemat składania działań.

16. Jeżeli zmienne x_1, x_2, \dots będą przyjmować wartości $w(x_1), w(x_2), \dots$ zaś symbole ze zbioru D będziemy uważać za działania n -argumentowe ($n=0, 1, \dots$) o wartościach ze zbioru X , to term $f(x_1, \dots, x_n)$ przyporządkowuje elementom x_1, \dots, x_n ze zbioru X element $y=f(x_1, \dots, x_n)$ z tego samego zbioru. Znając dany term i wiedząc, jakie działania w nim występują, będziemy wiedzieli, w jaki sposób została otrzymana wartość działania opisana tym termem, czyli jak interpretujemy term.

17. Algebrą abstrakcyjną (algebrą) będziemy nazywać każdy system $A=(K, O_1, O_2, \dots, O_n)$, w którym A jest niepustym zbiorem oraz $O_i, i=1, 2, \dots, n$ jest operacją i -argumentową w zbiorze K .

Literatura

- [1] Majewski M., Albicki A., Algebraiczna teoria automatów, WNT, Warszawa 1980,
 [2] Bromirski J., Teoria automatów, WNT, Warszawa 1969,
 [3] Marciszewski W. (red), Logika formalna. Zarys encyklopedyczny z zastosowaniem do informatyki i lingwistyki, PWN, Warszawa 1987.

2 PODSTAWY ALGEBRY BOOLE'A

W 1854 roku angielski matematyk George Boole zaproponował system algebraiczny nazywany obecnie algebrą Boole'a. Jej wykorzystanie do analizowania i opisu zachowania się układów zbudowanych z przełączników zaproponował dopiero w 1938 roku amerykański badacz Claude E. Shannon[1]. Algebra Shannona nazywana jest algebrą układów przełączających (*switching circuits*) dlatego, że przełącznik stosowany wtedy jako element budowy układów logicznych - pozwalał na przełączanie się między stanami przewodzenia albo nieprzewodzenia, co modelowało się za pomocą zmiennej x przyjmującej dwie wartości: 0 albo 1.

2.1 Podstawy algebry Boole'a

Do zdefiniowania algebry Boole'a wykorzystamy zestaw postulatów podany w roku 1904 przez E.V. Huntingtona[2]. Postulaty Huntingtona nie są jedynymi, za pomocą których zdefiniowano algebrę Boole'a - tutaj o wyborze zdecydowała minimalna ich liczba oraz względy dydaktyczne, przede wszystkim związane z twierdzeniem o asocjatywności, które dowodzi się wykorzystując jedynie postulaty.

Dowolną algebrę charakteryzujemy przez podanie zbioru rozważanych obiektów, elementów wyróżnionych w tym zbiorze, operacji określonych w zbiorze obiektów oraz pewnych relacji zachodzących między nimi, a także postulatów charakteryzujących operacje i relacje.

Jeżeli przyjmiemy, że K to co najmniej dwuelementowy zbiór obiektów, oraz

- $0, 1 \in K$ - operatory zeroargumentowe (elementy wyróżnione),
- $\bar{} : K \rightarrow K$ - operator jednoargumentowy (unarny),
- $+, \cdot : K \times K \rightarrow K$ - operatory dwuargumentowe (binarne),
- $= : K \times K \rightarrow \{\text{prawda}, \text{fałsz}\}$ - relacja równości,

to siódmka $AB=(K, 0, 1, +, \cdot, \bar{}, =)$ stanowi algebrę Boole'a, jeśli spełnione są następujące postulaty:

Postulat	a)	b)
1	$(\forall x, y \in K) (x + y \in K)$	$x \cdot y \in K$
2	$(\exists 0 \in K)(\forall x \in K)(x + 0 = x)$	$(\exists 1 \in K)(\forall x \in K)(x \cdot 1 = x)$
3	$(\forall x, y \in K) (x + y = y + x)$	$x \cdot y = y \cdot x$
4	$(\forall x, y, z \in K) (x + (y \cdot z) = (x + y) \cdot (x + z))$	$x \cdot (y + z) = (x \cdot y) + (x \cdot z)$
5	$(\forall x \in K) (\exists \bar{x} \in K)(x + \bar{x} = 1 \wedge x \cdot \bar{x} = 0)$	
6	Istnieją co najmniej dwa elementy $(x, y \in K)(x \neq y)$	

Znak $=$ oznacza, że wyrażenia stojące po lewej i prawej jego stronie są sobie równe. Nawiasy służą do grupowania wyrażeń i spełniają istotną rolę, szczególnie w postulatcie 4.

Postulaty 1÷4 zawierają pary wyrażeń oznaczone a) oraz b), które nazywamy dualnymi. Zasada dualności obowiązująca w algebrze Boole'a stwierdza, że jeżeli pewne wyrażenie jest prawdziwe, to wyrażenie dualne też jest prawdziwe. Wyrażenie dualne otrzymujemy poprzez zastąpienie wszystkich operacji $+$ przez \cdot , wszystkich operacji \cdot przez $+$, a także wszystkich 1 na 0 i 0 na 1.

Postulat 1, nazywany postulattem zamkniętości ze względu na operację $+$ (1a) oraz operację \cdot (1b), mówi, że wynik operacji należy także do zbioru K . Postulat 2 jest nazywany postulattem o istnieniu dwóch elementów neutralnych: 0 - element neutralny względem operacji $+$ oraz 1 - element neutralny względem operacji \cdot (elementy neutralne przyjęło się oznaczać jako 0 (1) ale tutaj one nie muszą oznaczać wartości 0 (1)). Postulat 3 mówi o regułach

przemienności (komutatywności) operacji dodawania (3a) i mnożenia (3b). Postulat 4 to postulat o rozłączności (dystrybutywności) operacji dodawania względem mnożenia (4a) oraz rozłączności mnożenia względem dodawania (4b). Postulat 5 wprowadza istnienie dopełnienia dla dowolnego elementu zbioru K. Ostatni postulat mówi, że zbiór K powinien zawierać dwa lub więcej różnych elementów.

Aby zwrócić uwagę na różnice pomiędzy algebrą Boole'a a dobrze nam znaną algebrą liczb rzeczywistych, przytoczymy niektóre z nich, szczególnie te, które zazwyczaj prowadzą do nieporozumień:

- zbiór K musi mieć co najmniej dwa różne elementy,
- w algebrze Boole'a nie istnieje operacja odejmowania ani dzielenia (w sensie operacji odwrotnej do $+$, \cdot),
- operator dopełnienia nie istnieje w algebrze liczb rzeczywistych,
- w algebrze liczb rzeczywistych nie zachodzi prawo $x + (y \cdot z) = (x + y) \cdot (x + z)$, zachodzi tylko prawo rozdzielności mnożenia względem dodawania,
- w algebrze Boole'a nie obowiązuje zasada skracania, czyli z równości $x \cdot y = x \cdot z$ nie wynika, że $y = z$, a także z $x + y = x + z$ nie wynika, że $y = z$ (zob. lemat 7).
- wśród postulatów nie ma prawa łączności, np. dla dodawania $(a+b)+c=a+(b+c)$. Prawo to zachodzi w algebrze Boole'a i można je udowodnić korzystając jedynie z postulatów (w niektórych podręcznikach prawo łączności włączone jest do zestawu postulatów).

2.2 Podstawowe własności algebry Boole'a

Podobnie jak w algebrze liczb rzeczywistych, oprócz zbioru postulatów podaje się podstawowe lematy i twierdzenia, ułatwiające wyrażanie wybranych własności algebry. Oczywiście, podstawowe lematy i twierdzenia wyprowadza się z postulatów.

Lemat	a)	b)
1	Element 0 jest jedyny	Element 1 jest jedyny
2 (idempotentność)	$(\forall x \in K)(x + x = x)$	$(\forall x \in K)(x \cdot x = x)$
3 (elementy neutralne)	$(\forall x \in K)(x + 1 = 1)$	$(\forall x \in K)(x \cdot 0 = 0)$
4	$\bar{0} = 1$	$\bar{1} = 0$
5	Każdy element ma jedyne dopełnienie.	
6 (absorpcja, pokrycie)	$(\forall x, y \in K)(x + (x \cdot y) = x)$	$(\forall x, y \in K)(x \cdot (x + y) = x)$
7	$(\forall x, y \in K)(x \cdot y = y \wedge x + y = y \Rightarrow x = y)$	
8 (inwolucja)	$(\forall x \in K)(\bar{\bar{x}} = x)$	
9	$(\forall x, y \in K)((x \cdot y) + (x \cdot \bar{y}) = x)$	$(x + y) \cdot (x + \bar{y}) = x$

Twierdzenie	a)	b)
10 (asocjatywność)	$(\forall x, y, z \in K)(x + (y + z) = (x + y) + z)$	$x \cdot (y \cdot z) = (x \cdot y) \cdot z$
11 (prawa de Morgana)	$(\forall x, y \in K)(\overline{x + y} = \bar{x} \cdot \bar{y})$	$\overline{x \cdot y} = \bar{x} + \bar{y}$
12 (konsensus)	$(\forall x, y, z \in K)(x \cdot y + \bar{x} \cdot z + y \cdot z = x \cdot y + \bar{x} \cdot z)$	$(x + y) \cdot (\bar{x} + z) \cdot (y + z) = (x + y) \cdot (\bar{x} + z)$
13	$(\forall x, y, z \in K)(x \cdot y + x \cdot \bar{y} \cdot z = x \cdot y + x \cdot z)$	$(x + y) \cdot (x + \bar{y} + z) = (x + y) \cdot (x + z)$
14	$(\forall x, y \in K)(x + (\bar{x} \cdot y) = x + y)$	$x \cdot (\bar{x} + y) = x \cdot y$

Aby uniknąć zbyt dużej liczby nawiasów, twierdzenia 12 oraz 13 sformułowaliśmy uwzględniając twierdzenie 10. Których nawiasów brakuje?

Wspomniana wyżej zasada dualności dotyczy także lematów, ponieważ wiele z nich zawiera lemat dualny b). Jak zażartował jeden z autorów, zasada dualności jest bardzo ważna z punktu widzenia studenta, któremu wystarczy „zapamiętanie” jedynie połowy wyrażień.

Drugą połowę (dualnych) wyrażeń otrzymujemy stosując zasadę dualności. Zasada dualności jest prosta, ale może prowadzić do otrzymania nieprawdziwych wyrażeń, jeżeli nie uwzględnimy nawiasów w nich występujących. Nawiasy wyznaczają kolejność wykonywania operacji. Po starannym przeanalizowaniu postulatów oraz lematów algebry Boole'a, dochodzimy do wniosku, że większy priorytet ma operacja (\cdot) aniżeli $(+)$. Dlatego wyrażenie $x + (x \cdot y) = x$ (lemat 6) moglibyśmy napisać w postaci $x + x \cdot y = x$. Łatwo jednak sprawdzić, że zastosowanie zasady dualności do $x + x \cdot y = x$ doprowadzi nas do wyrażenia nieprawdziwego: $x+y=x$ (Dlaczego?).

Dowody lematów i twierdzeń należy przeprowadzić w ramach ćwiczeń, wykorzystując jedynie postulaty Huntingtona. Wyznacznikiem bardzo dobrej znajomości postulatów algebry Boole'a jest poprawnie przeprowadzony dowód twierdzenia 10.

Poniżej, celem zilustrowania zalecanego sposobu dowodzenia, przytoczymy dowód lematu 2a i praw de Morgana.

Przykład:

Podaj dowód lematu 2a, czyli $x+x=x$.

Dowód:

$$\begin{aligned} X+x &= (x+x) \cdot 1 && \text{postulat 2b i 1a} \\ &= (x+x) \cdot (x+\bar{x}) && \text{postulat 5} \\ &= x + (x \cdot \bar{x}) && \text{Postulat 4a} \\ &= x+0 && \text{postulat 5} \\ &= x && \text{Postulat 2a} \end{aligned}$$

c. n. p.

Przykład:

Dowód konsensusu (tw. 12) przeprowadzimy zakładając, że udowodnione mamy twierdzenie o asocjatywności dodawania i mnożenia.

Dowód:

Mamy udowodnić, że $ab + \bar{a}c + bc = ab + \bar{a}c$. Tym razem rozpoczniemy od prawej strony równości:

$$\begin{aligned} ab + \bar{a}c &= ab1 + \bar{a}c1 && \text{Tw. 10, postulat 2b} \\ &= ab(1+c) + \bar{a}c(1+b) && \text{Tw. 10, Lemat 3b} \\ &= ab1 + \bar{a}c1 + abc + \bar{a}bc && \text{Postulat 3a, Tw. 10} \\ &= ab + \bar{a}c + bc && \text{Postulat 2b, 3b, Lemat 9a} \end{aligned}$$

c. n. p.

Przykład:

Podaj dowód pierwszego z praw de Morgana, $\overline{x+y} = \bar{x} \cdot \bar{y}$. Aby nie przedłużać dowodu, wykorzystamy niektóre z podanych lematów. Pamiętajmy jednak, że lematy te należy udowodnić wykorzystując jedynie postulaty Huntingtona.

Dowód:

Oznaczmy przez $A=x+y$, a przez $B = \bar{x} \cdot \bar{y}$. Nasze rozumowanie sprowadzamy do pokazania prawdziwości następującego układu równań:

$$\begin{aligned} A+B &= 1 \\ A \cdot B &= 0 \end{aligned}$$

ponieważ z postulatu 5 i lematu 5 moglibyśmy wywnioskować, że

$$x+y = \overline{\bar{x} \cdot \bar{y}} \quad \text{oraz} \quad \overline{x+y} = \overline{\bar{x} \cdot \bar{y}}$$

i dalej na podstawie lematu 8

$$\overline{x+y} = \bar{x} \cdot \bar{y}.$$

Pokażemy zatem, że

$$A + B = (x + y) + (\bar{x} \cdot \bar{y}) = 1$$

oraz $A \cdot B = (x + y) \cdot (\bar{x} \cdot \bar{y}) = 0$

$$\begin{aligned} A+B &= (x + y) + (\bar{x} \cdot \bar{y}) = [(x + y) + \bar{x}] \cdot [(x + y) + \bar{y}] = && \text{postulat 4} \\ &= [x + (y + \bar{x})] \cdot [x + (y + \bar{y})] = && \text{twierdzenie 10} \\ &= [x + (\bar{x} + y)] \cdot [x + (y + \bar{y})] = && \text{postulat 3} \\ &= [(x + \bar{x}) + y] \cdot [x + (y + \bar{y})] = && \text{twierdzenie 10} \\ &= (1 + y) \cdot (x + 1) = && \text{postulat 5} \\ &= (y + 1) \cdot (x + 1) = && \text{postulat 3} \\ &= 1 \cdot 1 = && \text{lemat 3} \\ &= 1 \end{aligned}$$

Analogicznie postępując możemy pokazać, że

$$\begin{aligned} A \cdot B &= (x + y) \cdot (\bar{x} \cdot \bar{y}) = (\bar{x} \cdot \bar{y}) \cdot (x + y) \\ &= (\bar{x} \cdot \bar{y}) \cdot x + (\bar{x} \cdot \bar{y}) \cdot y \\ &= (\bar{x} \cdot x) \cdot \bar{y} + \bar{x} \cdot (\bar{y} \cdot y) \\ &= (0 \cdot \bar{y}) + (\bar{x} \cdot 0) \\ &= 0 \cdot 0 \\ &= 0 \end{aligned}$$

Celowo pominęliśmy niektóre nawiasy i wskazanie wykorzystywanych postulatów czy lematów, traktując tę część dowodu jako ćwiczenie.

Drugie prawo de Morgana $\overline{x \cdot y} = \bar{x} + \bar{y}$ można udowodnić wykorzystując pierwsze prawo de Morgana, ponieważ $\bar{x} + \bar{y} = \overline{\overline{\bar{x} + \bar{y}}} = \overline{\overline{\bar{x}} \cdot \overline{\bar{y}}} = \overline{x \cdot y}$. c. n. p.

Przykład:

Niech będzie dany czteroelementowy zbiór $K=\{1,2,5,10\}$ oraz działania $+, \cdot, \bar{}$ zdefiniowane w sposób następujący:

$$(\forall x, y \in K) (x + y = \text{najmniejsza wspólna wielokrotność liczb } x \text{ i } y)$$

$$(\forall x, y \in K) (x \cdot y = \text{największy wspólny dzielnik liczb } x \text{ i } y)$$

$$(\forall x \in K) (\bar{x} = 10/x)$$

Wykażemy, że czwórka $(\{1,2,5,10\}, +, \cdot, \bar{})$ stanowi algebrę Boole'a. Na rysunku 2.1 przedstawiliśmy tabelki działań, wypełnionych zgodnie z powyższą definicją.

+	1	2	5	10
1	1	2	5	10
2	2	2	10	10
5	5	10	5	10
10	10	10	10	10

•	1	2	5	10
1	1	1	1	1
2	1	2	1	2
5	1	1	5	5
10	1	2	5	10

x	\bar{x}
1	10
2	5
5	2
10	1

Rys. 2.1 Definicja działań w przykładowej algebrze Boole'a

Postulat 1 jest spełniony, ponieważ wynik działania należy do zbioru K. Istnieje element neutralny dla operacji dodawania, którym jest $x=1$ (postulat 2a) oraz mnożenia $x=10$ (postulat 2b).

Postulat 3 jest spełniony, ponieważ wartości położone symetrycznie względem przekątnej tablicy są sobie równe.

Postulat 4 jest spełniony, jeżeli dla dowolnych trzech wartości $x, y, z \in K$, $x+(y \cdot z) = (x+y) \cdot (x+z)$ oraz $x \cdot (y+z) = (x \cdot y) + (x \cdot z)$. W naszym przypadku różnych trójek jest 64, dlatego pokażemy przykładową, spełniającą postulat 4a:

$$\text{lewa strona: } 1 + (2 \cdot 5) = 1 + 1 = 1$$

$$\text{prawa strona: } (1 + 2) \cdot (1 + 5) = 2 \cdot 5 = 1$$

Postulat 5 jest spełniony, co łatwo zauważyć z rys. 2.1. Z oczywistego względu spełniony jest także postulat 6.

2.3 Dwuelementowa algebra Boole'a

Definiując algebrę Boole'a na zbiorze obiektów K powiedzieliśmy, że obiektów tych powinno być co najmniej dwa. W tym rozdziale przyjmujemy $K=\{0,1\}$ oraz zdefiniujemy trzy podstawowe operacje $\cdot, +$ oraz $\bar{}$ zgodnie z rys. 2.2.

Wykażemy, że czwórka $(\{0,1\}, \cdot, +, \bar{})$ stanowi dwuelementową algebrę Boole'a. W tym celu wystarczy pokazać, że postulaty Huntingtona są spełnione dla zdefiniowanych operacji na zbiorze $K=\{0,1\}$.

x	y	xy	x+y
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	1

Rys. 2.2 Definicja operacji mnożenia, dodawania i dopełnienia

x	\bar{x}
0	1
1	0

Zamkniętość operacji jest oczywista, ponieważ wynikiem każdej z nich jest element należący do zbioru K. Z definicji operacji wynika, że $0+0=0$, $0+1=1+0=1$ oraz $1+1=1$, $1 \cdot 0=0 \cdot 1=0$, czyli istnieją elementy neutralne, 0 dla operacji $+$ oraz 1 dla operacji \cdot . Spełnione jest prawo przemienności, ponieważ występuje symetria w układzie wartości elementów x oraz y w tabelach definiujących operacje \cdot oraz $+$ (rys. 2.2). Prawo dystrybutywności objaśnimy wykorzystując rys. 2.3 (tylko postulat 4b, czyli dystrybutywność operacji \cdot względem $+$).

Zauważmy, że kolumny 3 oraz 6 są identyczne. W analogiczny sposób możemy sprawdzić spełnienie postulat 4a. Z definicji operacji dopełnienia łatwo wykazać, że $x + \bar{x} = 1$, bo $0 + \bar{0} = 0 + 1 = 1$ oraz $1 + \bar{1} = 1 + 0 = 1$, a także $x \cdot \bar{x} = 0$, ponieważ $0 \cdot \bar{0} = 0 \cdot 1 = 0$ oraz $1 \cdot \bar{1} = 1 \cdot 0 = 0$. Postulat szósty jest spełniony, ponieważ $1 \neq 0$.

xyz	y+z	$x \cdot (y+z)$	$x \cdot y$	$x \cdot z$	$(x \cdot y) + (x \cdot z)$
1	2	3	4	5	6
000	0	0	0	0	0
001	1	0	0	0	0
010	1	0	0	0	0
011	1	0	0	0	0
100	0	0	0	0	0
101	1	1	0	1	1
110	1	1	1	0	1
111	1	1	1	1	1

Rys. 2.3 Prawo dystrybutywności

będzie równoważne

$$x + (yz) = (x + y)(x + z)$$

Następnie - uwzględniając własności wynikające z lematów - możemy napisać:

$$x + yz = (x + y)(x + z)$$

Przyjmijmy jako zasadę, że w wyrażeniach, operator mnożenia (\cdot) będziemy pomijać, jeżeli nie będzie to prowadzić do nieporozumień. Dlatego przykładowe wyrażenie postaci:

$$x + (y \cdot z) = (x + y) \cdot (x + z)$$

W dalszej części podręcznika będziemy się zajmować dwuelementową algebrą Boole'a, nazywaną także algebrą układów przełączających. C.E. Shannon pokazał, że algebra ta nadaje się do opisywania własności układów zawierających przekładniki elektromagnetyczne. Przekładnik ma dwa stany - przewodzenia albo nieprzewodzenia, które w naturalny sposób modeluje się za pomocą elementu 1 oraz elementu 0 (lub odwrotnie, zależnie od przyjętej konwencji). Ponieważ przejście z jednego stanu w drugi następuje na skutek „przełączenia”, stąd nazwa - układ przełączający. Nazwa ta przetrwała do dnia dzisiejszego, mimo że przekładniki nie są już praktycznie wykorzystywane. Ich rolę pełnią w układach logicznych elementy elektroniczne.

Na rysunku 2.4 zestawiliśmy odpowiedniości występujące między elementami i operacjami algebry Boole'a, a elementami i operatorami występującymi w teorii zbiorów i rachunku zdań.

Algebra Boole'a	Teoria zbiorów	Rachunek zdań
Elementy	Rodzina podzbiorów zbioru zamkniętego na operacje $\cap \cup -$	
zdanie logiczne		
\cdot	przecięcie \cap	koniunkcja
$+$	suma mnogościowa \cup	alternatywa
Dopełnienie $\bar{}$	dopełnienie zbioru	negacja
Element 0	zbiór pusty \emptyset	<u>falsz</u>
Element 1	przestrzeń	<u>prawda</u>

Rys. 2.4 Związek algebry Boole'a z teorią zbiorów i rachunkiem zdań

Literatura

- [1] Shannon C.E., A Symbolic Analysis of Relay and Switching Circuits, Trans. AIEE, Vol. 57, 1938, str. 713-723,
 [2] Huntington E.V., Postulates for the Algebra of Logic, Trans. Am. Math. Soc., vol.5, 1904, str. 288-309

ĆWICZENIA

- Mamy dane operacje nad skończonym zbiorem $K \in \{0,1\}$: $a+b=\max(a,b)$, $a \cdot b=\min(a,b)$, $\bar{0}=1$ oraz $\bar{1}=0$. Wykazać, że siódemka $(K,=,+,\cdot,\bar{},1,0)$ jest dwuelementową algebrą Boole'a.
- Podaj postulaty i lematy algebry Boole'a, jeżeli operatory boolowskie będą miały następującą interpretację arytmetyczną

$$x \wedge y = x \cdot y$$

$$x \vee y = (x+y) - (x \cdot y)$$

$$\bar{x} = 1 - x,$$
 gdzie:

 $\wedge, \vee, \bar{}$ - operatory boolowskie: mnożenie, dodawanie i dopełnienie,

 $\cdot, +, -$ - operatory arytmetyczne: mnożenie, dodawanie i odejmowanie.
- Czy istnieje trzelementowa algebra Boole'a ?
- Udowodnić lematy i twierdzenia algebry Boole'a, korzystając jedynie z postulatów Huntingtona lub z wcześniej udowodnionych lematów bądź twierdzeń.
- Wykazać, że $x+x+\dots+x=x$ oraz $x \cdot x \cdot \dots \cdot x = x$ korzystając jedynie z postulatów.

6. Wykazać, że $\overline{x_1 \cdot x_2 \cdot \dots \cdot x_n} = \bar{x}_1 + \bar{x}_2 + \dots + \bar{x}_n$ korzystając z lematów i twierdzeń.

7. Czy prawdą jest, że:

a) $a \cdot (b+c) \cdot (d+e) = a \cdot b \cdot d + a \cdot b \cdot e + a \cdot c \cdot d + a \cdot c \cdot e$

b) $(abc) + (de) = (a+d)(a+e)(b+d)(b+e)(c+d)(c+e)$

9. Podaj dopełnienie każdego z poniższych wyrażeń:

a) $x + (y \cdot z)$

b) $x \cdot (y + z \cdot (w + \bar{x}))$

c) $x(y+z) + (\bar{x} \cdot y)$.

10. Zdefiniujmy relację \subseteq w następujący sposób: $a \subseteq b \Leftrightarrow a \cdot \bar{b} = 0$. Przyjmując, że x, y, z są elementami algebry Boole'a, udowodnij prawdziwość następujących stwierdzeń:

a) $(x \subseteq y) \wedge (y \subseteq z) \Rightarrow (x \subseteq z)$

b) $(x \subseteq y) \wedge (x \subseteq z) \Rightarrow (x \subseteq y \cdot z)$

c) $(x \subseteq y) \Rightarrow (\forall z)(x \subseteq y+z)$

d) $(x \subseteq y) \Leftrightarrow (\bar{y} \subseteq \bar{x})$

e) $0 \subseteq 1$

f) $(x \subseteq y) \wedge (y \subseteq x) \Rightarrow (x=y)$

Jeżeli relację \subseteq zdefiniowalibyśmy $a \subseteq b \Leftrightarrow a \cdot b = a$ to czy prawdziwe byłyby powyższe stwierdzenia?

11. Udowodnij, że dla elementów a oraz b algebry Boole'a spełniających warunek $a \subseteq b$, spełniona jest równość $a + (b \cdot c) = b \cdot (a + c)$ dla dowolnego c .

12. Pokazać, że gdy $(a \cdot b)(\bar{a} + \bar{b}) = 0$ oraz $(a \cdot b) + (\bar{a} + \bar{b}) = 1$ to $\overline{(a \cdot b)} = \bar{a} + \bar{b}$.

3 FUNKCJE BOOŁOWSKIE

Przez funkcję na zbiorze elementów X , o wartościach należących do zbioru Z , rozumiemy relację f o dziedzinie X i przeciwdziedzinie Z , a więc pewien podzbiór produktu $X \times Z$ taki, że:

1. $(\forall x \in X)(\exists z \in Z)((x, z) \in f)$
2. $((x, z_1) \in f \wedge (x, z_2) \in f) \Rightarrow z_1 = z_2$

Pierwszy element pary $(x, z) \in f$ nazywamy argumentem. Drugi element tej pary nazywamy wartością funkcji f dla argumentu x . Oznaczamy go przez $f(x)=z$. Często zamiast terminu funkcja f określona na zbiorze X o wartościach ze zbioru Z mówimy o odwzorowaniu zbioru X w zbiór Z .

3.1 Definicja funkcji boolowskiej

Przyjmijmy, że x_1, x_2, \dots, x_n są zmiennymi, nie precyzując na razie, jakie elementy można za te zmienne podstawić. Termem nazwiemy pewne wyrażenie, zbudowane ze znaków działań i zmiennych, przy czym wyrażenie to zdefiniujemy indukcyjnie:

1. termami są symbole x_1, x_2, \dots, x_n (symbole oznaczające zmienne) oraz symbole działań zeroargumentowych.
2. jeżeli $Ala_1(x_{a_1}, \dots, x_{a_k}), \dots, Ala_n(x_{b_1}, \dots, x_{b_r})$ są termami zmiennych podanych w nawiasie, oraz Janek (x_1, x_2, \dots, x_n) jest działaniem n -argumentowym, pochodzącym ze zbioru działań jednoargumentowych, dwuargumentowych, trójargumentowych, ... itd., to wyrażenie Janek($Ala_1(x_{a_1}, \dots, x_{a_k}), \dots, Ala_n(x_{b_1}, \dots, x_{b_r})$) jest termem.

Przyjmijmy, że działaniem zeroargumentowym nazywamy ustalony element zbioru K . Działanie to możemy uważać za funkcję, których wartości są stałe. Działania można ze sobą składać i otrzymywać nowe działania.

Jak należy interpretować termy? Otóż termy są napisami wyznaczającymi schemat kolejnego składania działań. Jeżeli zmienne x_1, x_2, \dots będą przyjmować wartości x_1, x_2, \dots zaś symbole działań zero-, jedno- i więcej argumentowych o wartościach ze zbioru X , to term $f(x_1, x_2, \dots, x_n)$ wyznaczy pewne działanie. Działanie to przyporządkowuje elementom x_1, x_2, \dots, x_n ze zbioru X element $y=f(x_1, x_2, \dots, x_n)$ z tego samego zbioru. Znając dany term i wiedząc, jakie działania w nim występują, będziemy wiedzieli, w jaki sposób zostało otrzymane działanie opisane tym termem. Termy służą więc do opisu działań.

Wprowadźmy teraz pojęcie wyrażenia boolowskiego. Zbiorem wyrażeń boolowskich nazywamy najmniejszy zbiór spełniający następujące warunki:

- a) zmienne x_1, x_2, \dots, x_n oraz działania zeroargumentowe 0 i 1,
- b) gdy $f(x_1, x_2, \dots, x_n)$ jest termem, wtedy termem jest $g(x_1 x_2, \dots, x_n) = \bar{f}(x_1 x_2, \dots, x_n)$,
- c) jeżeli $f(x_1, x_2, \dots, x_n)$ i $g(x_1, x_2, \dots, x_n)$ są termami, to
 $h(x_1, x_2, \dots, x_n) = f(x_1, x_2, \dots, x_n) + g(x_1, x_2, \dots, x_n)$ oraz
 $k(x_1 x_2, \dots, x_n) = f(x_1 x_2, \dots, x_n) \cdot g(x_1 x_2, \dots, x_n)$ są termami.

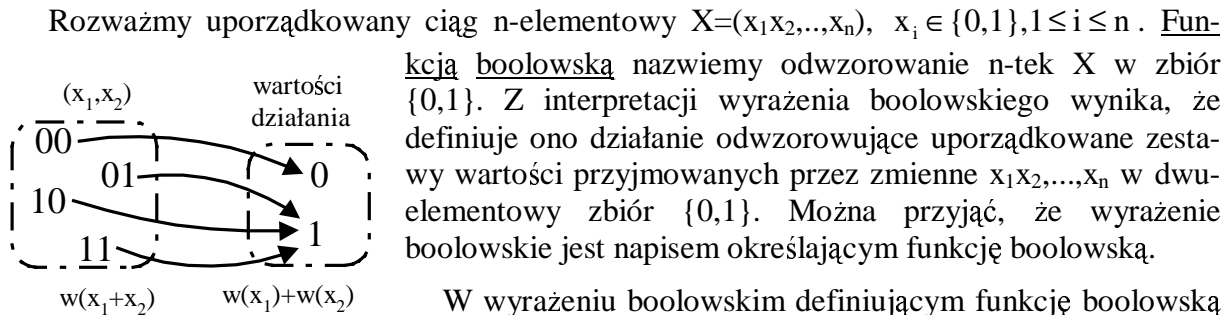
Przyjmijmy, że otrzymane termy mają następującą interpretację (przez $w(x)$ oznaczyliśmy wartość x).

1. $w(0) \rightarrow 0, w(1) \rightarrow 1$,
2. $w : \{x_1 x_2, \dots, x_n\} \rightarrow \{0, 1\}$. Zapis ten oznacza, że jeśli zmiennym $\{x_1 x_2, \dots, x_n\}$ przyporządkujemy $\{0, 1, \dots, 0\}$, to rozumiemy to jako przyporządkowanie zmiennej x_1 wartości 0, zmiennej x_2 wartości 1, ..., zmiennej x_n wartości 0.
3. $w(f + g) \rightarrow w(f) + w(g)$

4. $w(f \cdot g) \rightarrow w(f) \cdot w(g)$
5. $w(\bar{f}) \rightarrow \overline{w(f)}$

Przykład:

Przykładowe wyrażenie boolowskie postaci x_1+x_2 będzie miało następującą interpretację, zilustrowaną na rys. 3.1.

Rys. 3.1 Interpretacja termu x_1+x_2

W wyrażeniu boolowskim definiującym funkcję boolowską mogą występować dopełnienia zmiennych. Jednak w oznaczeniu funkcji, tzn. $f(x_1x_2,...,x_n)$, będziemy zaznaczać zmienne, które rozważamy. W dalszej części podręcznika pojęcia funkcja boolowska i funkcja przełączająca będziemy traktować jako synonimy.

3.2 Opisy funkcji boolowskiej

W poprzednim rozdziale podaliśmy definicję funkcji boolowskiej, wykorzystując odpowiednią konstrukcję wyrażenia, nazwanego wyrażeniem boolowskim. Wyrażenie boolowskie tworzone jest w skończonej liczbie kroków. Zatem w zależności od np. kolejności wyboru reguł definiujących wyrażenie boolowskie, powstaną różne wyrażenia (napisy), z których w rezultacie otrzymujemy to samo odwzorowanie. Wynika z tego potrzeba tworzenia pewnych standardowych postaci wyrażeń boolowskich, nazywanych także postaciami kanonicznymi (zbiór wyrażeń może być nieskończony, zaś zbiór postaci kanonicznych, jak się przekonamy, jest skończony).

3.2.1 Twierdzenie Shannona, postacie kanoniczne**Twierdzenie 3.1**

Dowolna funkcja jednej zmiennej x może być przedstawiona w postaci

$$f(x_1) = f(1) \cdot x_1 + f(0) \cdot \bar{x}_1$$

Twierdzenie 3.2

Dowolna funkcja może być przedstawiona w postaci

- a) $f(x_1x_2,...,x_n) = f(1, x_2,...,x_n) \cdot x_1 + f(0, x_2,...,x_n) \cdot \bar{x}_1$
- b) $f(x_1x_2,...,x_n) = [x_1 + f(0, x_2,...,x_n)] \cdot [\bar{x}_1 + f(1, x_2,...,x_n)]$

Dowód:

Każda zmienna $x_i, 1 \leq i \leq n$, może przyjmować wartości ze zbioru $\{0,1\}$. Niech $x_1=1$, wtedy dla postaci a) otrzymamy

$$f(1, x_2,...,x_n) = f(1, x_2,...,x_n) \cdot 1 + f(0, x_2,...,x_n) \cdot 0 = f(1, x_2,...,x_n).$$

Przyjmijmy teraz $x_1=0$. Wtedy

$$f(0, x_2,...,x_n) = f(1, x_2,...,x_n) \cdot 0 + f(0, x_2,...,x_n) \cdot 1 = f(0, x_2,...,x_n)$$

czyli otrzymaliśmy identyczność.

W analogiczny sposób można pokazać prawdziwość postaci b).

c.n.p.

Postać funkcji boolowskiej wynikającej z twierdzenia 3.2 nazywamy rozwinięciem funkcji ze względu na zmienną x_i . Twierdzenie 3.2 jest także prawdziwe dla dowolnej zmiennej x_i , $1 \leq i \leq n$. Rozwijając funkcje $f(1, x_2, \dots, x_n)$ oraz $f(0, x_2, \dots, x_n)$ ze względu na zmienną x_2 otrzymujemy następującą postać funkcji:

$$\begin{aligned} f(x_1 x_2, \dots, x_n) &= f(1, x_2, \dots, x_n) \cdot x_1 + f(0, x_2, \dots, x_n) \cdot \bar{x}_1 = \\ &[f(1, 1, \dots, x_n) \cdot x_2 + f(1, 0, \dots, x_n) \cdot \bar{x}_2] \cdot x_1 + [f(0, 1, \dots, x_n) \cdot x_2 + f(0, 0, \dots, x_n) \cdot \bar{x}_2] \cdot \bar{x}_1 = \\ &f(1, 1, \dots, x_n) \cdot x_1 x_2 + f(1, 0, \dots, x_n) \cdot x_1 \bar{x}_2 + f(0, 1, \dots, x_n) \cdot \bar{x}_1 x_2 + f(0, 0, \dots, x_n) \cdot \bar{x}_1 \bar{x}_2 \end{aligned}$$

Postępując jak wyżej, możemy analogicznie rozwinąć powyższą funkcję ze względu na x_3 , następnie x_4 , itd., przekonując się, że prawdziwe będzie następujące twierdzenie:

Twierdzenie 3.3 (Shannona)

Dowolną funkcję boolowską $f(x_1 x_2, \dots, x_n)$ można przedstawić w postaci

$$\begin{aligned} f(x_1, x_2, \dots, x_n) &= f(1, 1, 1, \dots, 1) x_1 x_2 x_3 \dots x_n + \\ &f(0, 1, 1, \dots, 1) \bar{x}_1 x_2 x_3 \dots x_n + \\ &f(1, 0, 1, \dots, 1) x_1 \bar{x}_2 x_3 \dots x_n + \\ &f(0, 0, 1, \dots, 1) \bar{x}_1 \bar{x}_2 x_3 \dots x_n + \\ &\dots \\ &f(0, 0, 0, \dots, 0) \bar{x}_1 \bar{x}_2 \bar{x}_3 \dots \bar{x}_n \end{aligned}$$

Postać funkcji podaną w twierdzeniu 3.3, powstałą w wyniku jej rozwinięcia ze względu na wszystkie zmienne, nazwiemy kanoniczną postacią dysjunkcyjną, zaś iloczyny n zmiennych (właściwie literałów) występujących w postaci kanonicznej - iloczynami zupełnymi (*minterm*). Funkcję z ustalonymi wartościami wybranych bądź wszystkich zmiennych będziemy nazywać funkcją resztową (*residual*) albo kofaktorem (*cofaktor*). Zwróćmy uwagę, że funkcja resztowa z ustalonymi wartościami wszystkich zmiennych jest funkcją stałą przyjmującą wartości 0 albo 1 (Dlaczego?).

Przykład:

Zgodnie z twierdzeniem 3.3 dowolną funkcję dwóch zmiennych $f(x_1, x_2)$ można rozwinąć do postaci: $f(x_1 x_2) = f(1, 1) \cdot x_1 x_2 + f(0, 1) \cdot \bar{x}_1 x_2 + f(1, 0) \cdot x_1 \bar{x}_2 + f(0, 0) \cdot \bar{x}_1 \bar{x}_2$. Zatem rozwinięcie przykładowej funkcji $f(x_1 x_2) = x_1$ do kanonicznej postaci dwóch zmiennych x_1, x_2 jest następujące:

$$\begin{aligned} f(x_1, x_2) &= f(1, 1) \cdot x_1 x_2 + f(0, 1) \cdot \bar{x}_1 x_2 + f(1, 0) \cdot x_1 \bar{x}_2 + f(0, 0) \cdot \bar{x}_1 \bar{x}_2 = \\ &= 1 \cdot x_1 x_2 + 0 \cdot \bar{x}_1 x_2 + 1 \cdot x_1 \bar{x}_2 + 0 \cdot \bar{x}_1 \bar{x}_2 = \\ &= x_1 x_2 + x_1 \bar{x}_2 \end{aligned}$$

Twierdzenie 3.4

Dowolną funkcję boolowską $f(x_1 x_2, \dots, x_n)$ można przedstawić w postaci

$$\begin{aligned} f(x_1 x_2, \dots, x_n) &= [f(1, 1, 1, \dots, 1) + \bar{x}_1 + \bar{x}_2 + \bar{x}_3 + \dots + \bar{x}_n] \cdot \\ &[f(0, 1, 1, \dots, 1) + x_1 + \bar{x}_2 + \bar{x}_3 + \dots + \bar{x}_n] \cdot \\ &[f(1, 0, 1, \dots, 1) + \bar{x}_1 + x_2 + \bar{x}_3 + \dots + \bar{x}_n] \cdot \\ &[f(0, 0, 1, \dots, 1) + x_1 + x_2 + \bar{x}_3 + \dots + \bar{x}_n] \cdot \\ &\dots \\ &[f(0, 0, 0, \dots, 0) + x_1 + x_2 + x_3 + \dots + x_n] \end{aligned}$$

Postać tę nazywamy kanoniczną postacią koniunkcyjną. Sumy zmiennych występujące w rozwinięciu nazywamy sumami zupełnymi (*maxterm*).

Przykład:

Kanoniczna postać koniunkcyjna dla funkcji z poprzedniego przykładu jest następująca:

$$\begin{aligned} f(x_1 x_2) &= [f(1,1) + \bar{x}_1 + \bar{x}_2] \cdot [f(0,1) + x_1 + \bar{x}_2] \cdot [f(1,0) + \bar{x}_1 + x_2] \cdot [f(0,0) + x_1 + x_2] = \\ &= [1 + \bar{x}_1 + \bar{x}_2] \cdot [0 + x_1 + \bar{x}_2] \cdot [1 + \bar{x}_1 + x_2] \cdot [0 + x_1 + x_2] = \\ &= (x_1 + \bar{x}_2) \cdot (x_1 + x_2) \end{aligned}$$

Z twierdzenia 3.3 i 3.4 wypływa następujący wniosek:

Wniosek 3.1

Istnieje jedyna kanoniczna postać dysjunkcyjna (koniunkcyjna) funkcji boolowskiej.

Dowód:

Patrz Harrison [1], s. 77.

Wynika z tego oczywisty wniosek:

Wniosek 3.2

Dwie funkcje boolowskie są równoważne wtedy i tylko wtedy, gdy mają takie same kanoniczne postacie dysjunkcyjne (koniunkcyjne).

3.2.2 Tablica prawdy i postać dziesiętna

Dowolną funkcję boolowską możemy przedstawić za pomocą wyrażenia boolowskiego. Kanoniczne postacie koniunkcyjne i dysjunkcyjne są specjalnymi wyrażeniami boolowskimi.

D	wartość zmiennych $x_1 \quad x_2$		wartość funkcji $f(x_1, x_2)$
1	2		3
0	0	0	0
1	0	1	0
2	1	0	1
3	1	1	1

Rys. 3.2 Tablica prawdy dla funkcji $f(x_1 x_2) = x_1$

Bardzo często funkcje boolowskie zapisywane są w postaci tablicy, nazwanej tablicą prawdy (*truth table*). Tablicę prawdy, dla przykładowej funkcji $f(x_1 x_2) = x_1$ przedstawiliśmy na rys. 3.2.

W kolumnie 3 wpisaliśmy wartości funkcji resztowych występujących w postaci dysjunkcyjnej (kanonicznej). Dla funkcji $f(x_1 x_2) = x_1$ otrzymujemy $f(1,1) = f(1,0) = 1$ oraz $f(0,0) = f(0,1) = 0$. W kolumnie 2 znajdują się wszystkie możliwe zestawy wartości dwóch zmiennych, a w kolumnie 1 ich odpowiednik dziesiętny, obliczony z następującego wzoru (operatory dodawania i mnożenia pochodzą z algebry liczb rzeczywistych):

$$D = \sum_{i=1}^n 2^{n-i} \cdot x_i \quad (3.1)$$

gdzie: n - liczba zmiennych

x_i - wartość i -tej zmiennej, $i = 1, 2, \dots, n$.

Wartości zmiennych występujące w kolumnie 2 na rys. 3.2 nazywane są notacją binarną lub szerzej notacją pozycyjną.

Rozważając odpowiednik dziesiętny danego zestawu wartości zmiennych, musimy znać kolejność ich występowania. Przyjmijmy, że zmienna x_1 występować będzie na najstarszej pozycji (odpowiada jej największa potęga dwójki we wzorze 3.1), a x_n na pozycji najmłodszej. Tę konwencję utrzymamy w całym podręczniku.

Odpowiedniki dziesiętne wykorzystujemy do dziesiętnego zapisu postaci dysjunkcyjnej (koniunkcyjnej). Dla funkcji $f(x_1 x_2) = x_1$, dziesiętna postać jest następująca: $f(x_1 x_2) = \sum(2,3)$. Znak sumy wskazuje, że jest to suma iloczynów zupełnych, których odpowiedniki dziesiętne

występują w nawiasach półokrągłych. Dla funkcji $f(x_1x_2)=x_1$ dziesiętna postać odpowiadająca kanonicznej postaci koniunkcyjnej jest następująca: $f(x_1x_2)=\prod(0,1)$. Znak iloczynu oznacza, że jest to iloczyn sum zupełnych, których odpowiedniki dziesiętne obliczono ze wzoru 3.1, dopełniając przedtem każdą składową rozważanego zestawu wartości (porównaj rys. 3.3).

Przykład:

Niech będzie dana funkcja $f(x_1x_2x_3)=x_1+x_2x_3$.

Związki zachodzące między poszczególnymi postaciami funkcji przełączającej przedstawiliśmy na rys. 3.3.

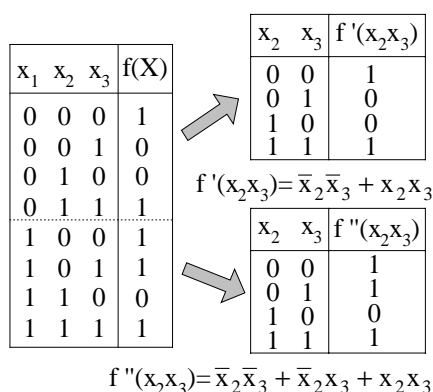
Postać koniunkcyjną lepiej stosować, gdy mniej jest zestawów wartości zmiennych, dla których funkcja przybiera wartość 0, ponieważ występuje wtedy mniej sum zupełnych niż iloczynów zupełnych w postaci dysjunkcyjnej. Jednak w dalszej części podręcznika będziemy wykorzystywać postać dysjunkcyjną.

Przykład:

Przedstawimy związki zachodzące pomiędzy tablicą prawdy a funkcjami resztowymi występującymi w rozwinięciu funkcji. Niech będzie dana funkcja $f(X) = x_2x_3 + x_1\bar{x}_2 + \bar{x}_2\bar{x}_3$. Posługując się tablicą prawdy, należy otrzymać funkcje resztowe f' i f'' , gdzie

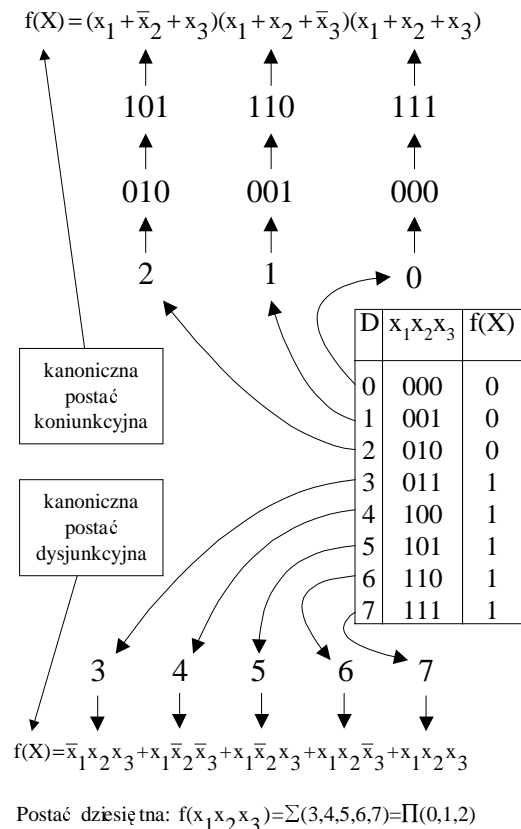
$$f(X) = \bar{x}_1 f(0, x_2, x_3) + x_1 f(1, x_2, x_3) = \bar{x}_1 f'(x_2, x_3) + x_1 f''(x_2, x_3)$$

(przykładowo rozwinęliśmy funkcję ze względu na zmienną x_1). Na rysunku 3.4 zilustrowaliśmy związki zachodzące między tablicą prawdy a funkcjami resztowymi f' oraz f'' .



Rys. 3.4 Związek tablicy prawdy z funkcjami resztowymi.

Boole'a.



Rys. 3.3 Związki zachodzące między różnymi postaciami zapisu funkcji przełączającej.

Funkcje resztowe (kofaktory), mają bardzo interesujące własności. Oznaczając $f(\dots, x_{i-1}, 0, \dots) = f_{\bar{x}_i}$ i $f(\dots, x_{i-1}, 1, \dots) = f_{x_i}$ proponujemy udowodnić następujące zależności:

$$(f \cdot g)_{x_i} = f_{x_i} \cdot g_{x_i}$$

$$(\bar{f})_{x_i} = \overline{f_{x_i}}$$

oraz najbardziej interesująca, pozwalająca obliczyć dopełnienie funkcji na podstawie kofaktorów:

$$\bar{f} = x_i \cdot \overline{f_{x_i}} + \bar{x}_i \cdot \overline{f_{\bar{x}_i}}$$

Poprawnie przeprowadzony dowód ostatniej własności jest sprawdzianem dobrego zrozumienia podstaw algebry

3.2.3 Funkcje przełączające nie w pełni określone

Na zakończenie rozważań o funkcjach przełączających wprowadzimy dodatkowe definicje.

Definicja 3.2.3.1

Funkcją przełączającą w pełni określoną będziemy nazywać funkcję, której wartość jest określona dla każdego zestawu wartości zmiennych.

Definicja 3.2.3.2

Funkcją przełączającą nie w pełni określoną będziemy nazywać funkcję, w której istnieją takie zestawy wartości zmiennych, dla których nie określono wartości funkcji.

Nieokreśloność wartości funkcji przełączającej należy rozumieć jako możliwość wyboru przez tę funkcję wartości 0 albo 1 dla ustalonego zestawu jej argumentów. Tak rozumianą nieokreśloność funkcji zaznaczamy $f(X)=\emptyset$, gdzie X jest iloczynem zupełnym nazywanym iloczynem obojętnym (*don't care*).

Nieokreśloność wartości funkcji może się zatem pojawić w dwóch przypadkach:

1. pewien zestaw iloczynów zupełnych nigdy nie wystąpi w postaci kanonicznej.
2. wszystkie iloczyny mogą definiować funkcję, ale dla pewnego ich podzbioru obojętna jest nam wartość funkcji. Jak się później okaże, iloczyny obojętne wykorzystamy przede wszystkim do minimalizacji funkcji przełączających.

W dalszej części podręcznika będziemy używać zamiennie następujących notacji na oznaczenie funkcji przełączającej: f , $f(X)$, $f(x_1, x_2, \dots, x_n)$, czy $f(x_1 x_2 \dots x_n)$.

3.2.4 Implikanty

Nie wnikając na razie w potrzebę wykorzystywania wyrażeń zawierających minimalną liczbę iloczynów w postaci dysjunkcyjnej, przytoczymy kilka definicji i twierdzeń umożliwiających „oszczędne” zapisanie dowolnej funkcji przełączającej. Znacznie więcej o metodach i potrzebie skracania (minimalizowania) wyrażenia boolowskiego dowiemy się w dalszej części podręcznika.

Wprowadźmy relację porządku \leq zdefiniowaną następująco:

$$(\forall x, y \in K)(x \leq y \Leftrightarrow x + y = y)$$

Relacja ta jest zwrotna (czyli $x \leq x$), antysymetryczna (jeżeli $x \leq y$ i $y \leq x$ to $x = y$) oraz przechodnia (jeżeli $x \leq y$ i $y \leq z$ to $x \leq z$), czyli jest relacją porządku. Można wykazać, że powyższa definicja porządku jest równoważna definicji: $x \leq y \Leftrightarrow x = x \cdot y$.

W dwuelementowej algebrze Boole’a spełniona jest relacja $0 \leq 1$. Dlatego, jeżeli dla dowolnych n -tek $A=(a_1, a_2, \dots, a_n)$ oraz $B=(b_1, b_2, \dots, b_n)$, $a_i, b_i \in \{0, 1\}$, $1 \leq i \leq n$ zachodzi relacja $a_i \leq b_i$, fakt ten zaznaczamy jako $A \leq B$.

Przyjmijmy, że $D_0(f)=\{X|f(X)=0\}$ oraz $D_1(f)=\{X|f(X)=1\}$. Wprowadźmy relację porządku \leq dla funkcji tych samych zmiennych postaci: $(\forall f, g)(f \leq g \Leftrightarrow f + g = g)$. Łatwo zauważyć, że $(f \leq g) \Leftrightarrow D_1(f) \subseteq D_1(g)$, który to związek zapiszemy w postaci definicji.

Definicja 3.2.4.1

Niech będą dane dwie funkcje przełączające $f(X)$ i $g(X)$. Jeżeli $(\forall X)(f(X)=1 \Rightarrow g(X)=1)$, to związek taki zapisujemy $f(X) \leq g(X)$ stwierdzając, że f implikuje funkcję g . Mówimy także, że g pokrywa f . Jeżeli $f \leq g$ lub $g \leq f$, to f oraz g nazywamy funkcjami porównywalnymi.

Dla przykładu funkcjami porównywalnymi są funkcje $f(x_1x_2)=x_1$ oraz $g(x_1x_2)=x_1+x_2$. Nie wszystkie funkcje są funkcjami porównywalnymi, np. $f(x_1x_2)=x_1$ oraz $g(x_1x_2)=x_1\bar{x}_2+\bar{x}_1x_2$.

Definicja 3.2.4.2

Implikantem t funkcji przełączającej f nazywamy iloczyn literałów (czyli zmiennych afirmowanych lub z dopełnieniem) implikujący funkcję f , czyli $t \leq f$.

Przykład:

Niech będzie dana następująca funkcja przełączająca: $f(x_1x_2x_3) = x_1 + x_2\bar{x}_3$. Implikantami funkcji są: $\bar{x}_1x_2\bar{x}_3$, $x_1x_2\bar{x}_3$, $x_1\bar{x}_2\bar{x}_3$, $x_1x_2x_3$, $x_1\bar{x}_2x_3$, x_1x_3 , $x_1\bar{x}_3$, $x_1\bar{x}_2$, $x_2\bar{x}_3$, x_1x_2 , x_1 . Zwróćmy uwagę na to, że x_1 jest traktowany tutaj jako iloczyn literałów. W celu ilustracji pokażemy, że na przykład iloczyn $x_1\bar{x}_2$ rzeczywiście jest implikantem powyższej funkcji, ponieważ powinna być spełniona równość: $x_1\bar{x}_2 + f = f$. Czyli $x_1\bar{x}_2 + f = x_1\bar{x}_2 + x_1 + x_2\bar{x}_3 = x_1(\bar{x}_2 + 1) + x_2\bar{x}_3 = x_1 + x_2\bar{x}_3 = f$.

Definicja 3.2.4.3

Implikantem prostym T nazywamy implikant, który po odrzuceniu z niego **dowolnego** literału przestaje być implikantem.

Przykład:

Niech będzie dana funkcja $f(x_1x_2x_3) = x_1x_3 + x_2\bar{x}_3$, która ma trzy implikanty proste - x_1x_3 , $x_2\bar{x}_3$ oraz x_1x_2 . W celu ilustracji pokażemy, że iloczyn x_1x_3 rzeczywiście jest implikantem prostym. Po odrzuceniu z niego x_3 pozostanie x_1 , czyli $x_1 + f \neq f$. Odrzucając literał x_1 , otrzymamy x_3 , czyli $x_3 + f \neq f$. Zatem zgodnie z definicją iloczyn x_1x_3 jest implikantem prostym. Przy okazji spróbujmy podać wszystkie implikanty proste dla funkcji $f(x_1x_2)=x_1+x_2$.

Twierdzenie 3.2.4.1

Każdy implikant funkcji przełączającej implikuje implikant prosty tej funkcji.

Dowód:

Jest oczywiste, że twierdzenie jest prawdziwe dla dowolnego implikantu, który jest implikantem prostym. Gdy implikant nie jest implikantem prostym, wtedy jeden lub więcej literałów może być z niego usuniętych, aż stanie się on implikantem prostym. Oznaczmy implikant prosty jako T_p , natomiast implikant jako iloczyn $T_p \cdot T_q$, gdzie T_q będzie iloczynem odrzuconych literałów. Stąd $T_p + T_p \cdot T_q = T_p$, czyli $T_q \leq T_p$. c.n.p.

Twierdzenie 3.2.4.2

Każda funkcja przełączająca może być wyrażona jako suma wyłącznie implikantów prostych.

Dowód:

Przedstawmy daną funkcję przełączającą f w postaci sumy iloczynów oraz przyjmijmy, że t_i jest iloczynem, który nie jest implikantem prostym. Przez F oznaczmy sumę pozostałych iloczynów. Wtedy $F \leq f$, ponieważ $f = F + t_i$. Załóżmy z kolei, że $t_i \leq P_i$, gdzie P_i jest implikantem prostym danej funkcji f . Niech $f_1 = F + P_i$. Ponieważ $t_i \leq P_i$ to $f \leq f_1$. Z drugiej strony P_i jest implikantem prostym funkcji f , więc $P_i \leq f$. Ponieważ $F \leq f$, więc $F + P_i \leq f$ lub $f_1 \leq f$. Wcześniej pokazaliśmy, że $f \leq f_1$, czyli $f_1 = f$. Wynika z tego, że wartość funkcji nie ulega zmianie w przypadku zastąpienia iloczynu t_i jego implikantem prostym (tzn. $t_i \leq P_i$).

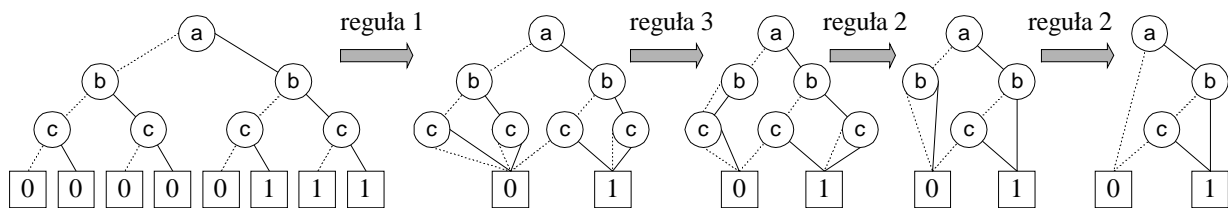
Podobnie wszystkie inne iloczyny mogą być zastąpione odpowiadającymi im implikantami prostymi. c.n.p.

W rozdziale 6 wprowadzimy dodatkowo pojęcie zasadniczego implikantu prostego (*essential prime implicant*).

3.2.5 Binarne diagramy decyzyjne.

Binarny diagram decyzyjny (w skrócie BDD) to acykliczny graf skierowany reprezentujący funkcję przełączającą.

Aby wyjaśnić sposób tworzenia BDD posłużymy się rys. 3.5, na którym przedstawiliśmy binarne drzewo decyzyjne (jeszcze nie diagram) dla funkcji $f(abc)=ab+\bar{a}\bar{b}c$. Gdy narysujemy tablice prawdy dla tej funkcji, to bez trudu zauważymy bezpośredni związek między tablicą a drzewem decyzyjnym. Z każdego wewnętrznego wierzchołka (nieterminalnego) oznaczonego nazwą zmiennej wychodzą dwie gałęzie oznaczone wartościami przyjmowanymi przez tę zmienną. Lewe gałęzie reprezentują wartość 0 (linie przerywane), a prawe wartość 1 (linie ciągłe). Liście drzewa (wierzchołki terminalne) reprezentują wartości funkcji, czyli 0 albo 1. Każda ścieżka od korzenia do liścia wraz z przypisanymi do krawędzi wartościami, odpowiada jednemu zestawowi wartości zmiennych występującemu w tablicy prawdy, a wartością funkcji dla danego zestawu jest wartość przypisana liściowi, do którego dotarliśmy.



Rys. 3.5 Kolejne kroki przekształcania binarnego drzewa decyzyjnego w uporządkowany ($a < b < c$) i zredukowany diagram decyzyjny reprezentujący funkcję $f(abc)=ab+\bar{a}\bar{b}c$

Drzewo binarne może mieć wiele nadmiarowych wierzchołków, które usuwamy. Oczywiście drzewo po ich usunięciu nadal powinno reprezentować tę samą funkcję.

Wyróżniamy trzy podstawowe reguły redukujące liczbę wierzchołków:

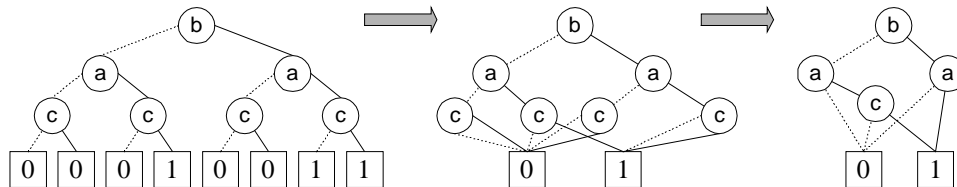
1. Łączymy ze sobą liście z tymi samymi wartościami (reguła 1 z rys. 3.5). Jest oczywiste, że po połączeniu liści z wartościami 0 i 1 drzewo nadal reprezentuje zadaną funkcję (dłaczego?).
2. Usuwamy wszystkie wierzchołki, z których obie wychodzące krawędzie dochodzą do tego samego wierzchołka (reguła 2 z rys. 3.5). Powyższe oznacza, że przechodząc od korzenia przez dany wierzchołek w kierunku liści, dojdziemy do potomka wierzchołka, bez względu na to, którą wartość przyjmie zmienna.
3. Łączymy wierzchołki z identycznymi poddrzewami (reguła 3 z rys. 3.5). Argumenty analogiczne jak w punkcie 2.

Uwaga: Nie musimy budować drzewa binarnego, aby otrzymać binarny diagram decyzyjny. Powyższa ilustracja ma jedynie pokazać związek między funkcją przełączającą a diagramem decyzyjnym.

Zwróćmy uwagę na to, że w drzewie jak i w zredukowanym grafie na dowolnej ścieżce od korzenia do liścia jest identyczne uporządkowanie zmiennych (dla przykładu na rys. 3.5 to uporządkowanie jest postaci $a < b < c$). Bryant [2] pokazał, że tak uporządkowany i zredukowany graf (w skrócie *ROBDD*) w unikalny sposób reprezentuje zadaną funkcję, czyli pełni ana-

logiczną rolę jak kanoniczna postać wyrażenia boolowskiego, gdy chcemy porównać dwie funkcje.

Jeżeli jednak drzewo binarne reprezentujące tę samą funkcję ma inne uporządkowanie zmiennych, to zredukowany graf może się różnić od poprzedniego nie tylko uporządkowaniem zmiennych, ale przede wszystkim liczbą wierzchołków (porównaj z rys. 3.6). Zatem by uporządkowane i zredukowane diagramy decyzyjne mogły być wykorzystywane np. do porównania dwóch funkcji, to oba diagramy powinny mieć identycznie uporządkowane zmienne.



Rys. 3.6 Kolejne kroki przekształcania binarnego drzewa decyzyjnego w uporządkowany ($b < a < c$) i zredukowany diagram decyzyjny reprezentujący funkcję $f(abc) = ab + \bar{a}\bar{b}c$

Na diagramach decyzyjnych możemy wykonywać operacje. Aby je omówić przytoczymy formalną definicję uporządkowanego binarnego diagramu decyzyjnego [2]:

Definicja 3.2.5.1

Uporządkowany binarny diagram decyzyjny jest skierowanym grafem z korzeniem o zbiorze wierzchołków V . Każdy wierzchołek nie będący liściem ma następujące atrybuty: wskaźnik $index(v) \in \{1, 2, \dots, n\}$ do indeksów zmiennych wejściowej ze zbioru $\{x_1, x_2, \dots, x_n\}$ oraz dwóch potomków $low(v)$ oraz $high(v) \in V$. Atrybutem wierzchołka będącego liściem, jest wartość $value(v) \in \{0, 1\}$. Dla każdej pary wierzchołków $\{v, low(v)\}$ ($\{v, high(v)\}$) takiej, że żaden z wierzchołków nie jest liściem, jest spełniona nierówność $index(v) < index(low(v))$ ($index(v) < index(high(v))$).

Związek pomiędzy uporządkowanym binarnym diagramem decyzyjnym a funkcją przełączającą jest definiowany następująco:

Definicja 3.2.5.2

Diagram G z korzeniem v reprezentuje funkcję f_v określoną rekurencyjnie w następujący sposób:

1. jeżeli v jest liściem to, gdy $value(v)=1(0)$, wtedy $f_v=1(0)$,
2. jeżeli v jest wierzchołkiem wewnętrznym z $index(v)=i$, wtedy reprezentuje on funkcję:

$$f_v(x_1, \dots, x_n) = \bar{x}_i \cdot f_{low(v)}(x_1, \dots, x_n) + x_i \cdot f_{high(v)}(x_1, \dots, x_n)$$

x_i nazywamy zmienną decyzyjną dla wierzchołka v .

Definicja 3.2.5.3

Dwa uporządkowane diagramy G_1 i G_2 są izomorficzne, jeżeli istnieje taka równoważnościowa funkcja δ odwzorowująca wierzchołki G_1 na wierzchołki G_2 , że dla dowolnego wierzchołka v , gdy $\delta(v)=w$, to albo oba wierzchołki v i w są liśćmi z $value(v)=value(w)$ albo v i w są wierzchołkami wewnętrznymi z $index(v)=index(w)$ oraz $\delta(low(v))=low(w)$ i $\delta(high(v))=high(w)$.

Zwróćmy uwagę na to, że gdy graf reprezentujący funkcję zawiera jeden korzeń a potomkowie dowolnego nieterminalnego wierzchołka są różni, to powyższa definicja izomorfizmu oznacza, że korzeń G_1 musi odwzorowywać się w korzeń w G_2 , lewy potomek G_1 w lewego potomka G_2 i tak dalej, aż do liści.

Definicja 3.2.5.4

Uporządkowany diagram jest zredukowany, jeżeli:

- nie zawiera wierzchołków v takich, że $low(v)=high(v)$,
- nie zawiera różnych wierzchołków v i w takich, że podgrafy z korzeniami v i w są izomorficzne.

Nietrudno zauważyć, że zredukowany graf nadal reprezentuje daną funkcję.

Twierdzenie 3.2.5.1

Dowolna funkcja boolowska f posiada unikalny (z dokładnością do izomorfizmu) zredukowany i uporządkowany diagram decyzyjny (ROBDD) reprezentujący daną funkcję a każdy inny uporządkowany diagram decyzyjny (OBDD) reprezentujący funkcję f zawiera większą liczbę wierzchołków.

Dalej będziemy używać skrótu BDD na oznaczenie uporządkowanych i zredukowanych diagramów decyzyjnych.

Operacje wykonywane na diagramach.

Atrakcyjność diagramów z punktu widzenia „łatwości” z jaką mogą reprezentować funkcję przełączającą wynika z tego, że funkcja zależy przede wszystkim od liczby wszystkich możliwych ścieżek prowadzących od korzenia do liścia z wartością jeden a w znacznie mniejszym stopniu zależy od liczby wierzchołków diagramu. Wspomniana atrakcyjność nie miałaby żadnego znaczenia, gdyby nie można było wykonywać podstawowych operacji, że wymienimy tylko mnożenia, dodawania czy dopełniania funkcji.

W poniższych podrozdziałach przedstawimy szereg funkcji pozwalających redukować diagramy, wykonywać dowolną binarną operację na dwóch diagramach, wyznaczyć funkcję resztową (kofaktor), dokonać podstawienia jednej funkcji w miejsce wybranej zmiennej w drugiej funkcji (obie funkcje reprezentowane za pomocą diagramów), sprawdzić czy funkcja dla każdego zestawu wartości zmiennych przyjmuje wartość 1 (tautologia), czy wreszcie znaleźć przynajmniej jeden zestaw wartości zmiennych wejściowych, dla którego funkcja równa się jeden (spełnialność).

Wspomniane funkcje opiszemy za pomocą object pascala. Podstawową strukturą danych, na której oprzemy algorytmy jest reprezentacja wierzchołka diagramu. Przyjmujemy, że każdy wierzchołek jest reprezen-

towany przez typ rekordowy przedstawiony na rys. 3.7a, bez względu na to czy reprezentuje wierzchołki terminalne czy nieterminalne (są one różniane za pomocą wartości odpowiednich pól – porównaj rys.3.7b). Wierz-

a)

```
type tVertex = record
  Low, High : tVertex;
  Index      : 1..N+1;
  Val        : (0,1,X);
  Id         : integer;
  Mark       : boolean;
end;
```

b)

Pole wierzchołka v	Terminalny	Nieterminalny
Low	NIL	Low(v)
High	NIL	High(v)
Index	N+1	Index(v)
Val	Value(v)	X

Rys. 3.7 Struktura danych a) wierzchołka diagramu, b) wartości pól

chołek nieterminalny ma zawsze dwóch potomków, do których odwołujemy się za pomocą pól low i high. Pole Index niesie informacje o indeksie zmiennej, a w wierzchołku terminal-

nym przyjmuje wartość $N+1$, gdzie N – liczba zmiennych. Dla wierzchołków terminalnych w polu *val* znajdują się wartości 0 albo 1, a dla nieterminalnych wartość pola jest nieokreślona.

Dwa pozostałe pola, *Id* oraz *Mark*, mają znaczenie organizacyjne wykorzystywane w opisywanych niżej algorytmach. *Id* zawiera zawsze wartość typu integer w unikalny sposób określającą wierzchołek, zaś pole *Mark* wspomaga proces przechodzenia po diagramie (odwiedzania wszystkich jego wierzchołków). Poniższa rekurencyjna procedura *PrzegladaajDrzewo* ilustruje wykorzystania pola *Mark* oraz ustawienie unikalnej wartości w polu *Id* (na podstawie zwiększania wartości pewnej zmiennej globalnej *Licznik*, gdy odwiedzimy wierzchołek).

```

Procedure PrzegladaajDrzewo(v : tVertex);
Begin
  v.mark := not v.mark;
  v.id := inc(Licznik);
  if v.index <= N then begin // v jest nieterminalny
    if v.mark <> v.low.mark then PrzegladaajDrzewo(v.low);
    if v.mark <> v.high.mark then PrzegladaajDrzewo(v.high);
  end;
end;

```

Procedura REDUCE.

Algorytm redukowania przekształca uporządkowany binarny diagram w jego zredukowany odpowiednik reprezentujący tę samą funkcję. Przypomnijmy, że diagram jest zredukowany, jeśli nie zawiera ani żadnego wierzchołka v , dla którego $\text{low}(v)=\text{high}(v)$, ani żadnej pary wierzchołków $\{u,v\}$ takiej, że podgrafy o korzeniach v i u są izomorficzne.

Istota algorytmu sprowadza się do odpowiedniego oznakowania wierzchołków uporządkowanego diagramu podczas przechodzenia przez graf. Wierzchołki terminalne mają te same etykiety (identyfikatory), jeżeli mają tę samą wartość pola *value*. Rozważmy podzbiór wszystkich wierzchołków o indeksie k , oznaczając go $W(k)$, czyli zawiera on wierzchołki związane z k -tą zmienną funkcji (mówimy także, że wierzchołki te znajdują się na i -tym poziomie). Do generowania podzbiorów $W(k)$ możemy wykorzystać nieco zmienioną procedurę *PrzegladaajDrzewo*. Jeżeli $\text{id}(\text{low}(v)) = \text{id}(\text{high}(v))$, to wierzchołek $v \in W(k)$ jest nadmiarowy. Wykonuje się wówczas podstawienie $\text{id}(v) := \text{id}(\text{low}(v))$. Podobnie, jeżeli istnieją dwa wierzchołki $u, v \in W(k)$ takie, że $\text{id}(\text{low}(v)) = \text{id}(\text{low}(u))$ oraz $\text{id}(\text{high}(v)) = \text{id}(\text{high}(u))$, to są one korzeniami dwóch grafów izomorficznych. Wykonujemy wówczas podstawienia $\text{id}(v) := \text{id}(u)$. W pozostałych przypadkach, każdemu wierzchołkowi z poziomu k nadajemy różne identyfikatory. Algorytm kończy działanie, gdy wykonując powyższe kroki, przechodząc z poziomu na poziom dotrzemy do korzenia.

Zamiast brania po uwagę wszystkich par wierzchołków danego poziomu i sprawdzania czy mają takich samych potomków, tworzymy klucz na podstawie tych identyfikatorów. Następnie sortujemy elementy zbioru $W(k)$ według klucza. Gdy klucz rozważanego wierzchołka jest równy kluczowi któregoś z już rozważanych wierzchołków, to usuwamy ten wierzchołek. W przeciwnym przypadku rozważanemu wierzchołkowi przypisuje się kolejną unikalną wartość (np. poprzez zwiększenie zawartości pewnej zmiennej globalnej) i umieszczamy go w zredukowanym diagramie.

Przykład:

Niech będzie dany diagram przedstawiony na rys. 3.8 (aby nie przedłużać przykładu połączymy jednakowe liście, którym nadaliśmy identyfikatory 1 oraz 2). Wierzchołki z jednokowym indeksem zgromadziliśmy na tych samych poziomach. Będziemy się poruszać po diagramie począwszy od poziomu liści ($\text{index}=5$) w kierunku korzenia ($\text{index}=1$) etykietując

wierzchołki odpowiednim kluczem. Liściom przyporządkowaliśmy klucz (0) oraz (1). Obok diagramu w tabeli wpisujemy wszystkie unikalne klucze w porządku rosnącym, zaś każdemu kluczowi przyporządkowujemy unikalny identyfikator. Z prawej strony tabelki będziemy rysować kolejne fazy powstawania zredukowanego diagramu.

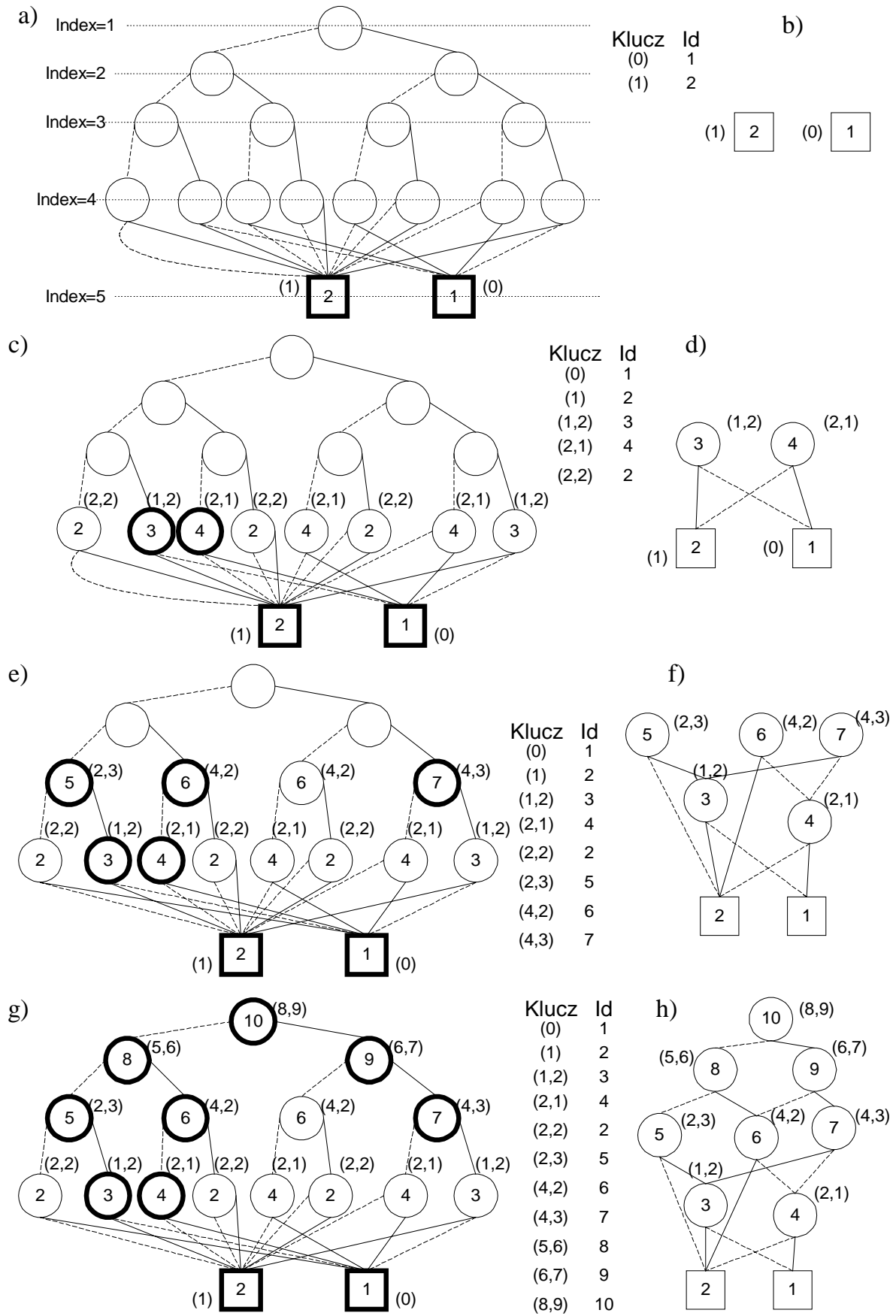
Ponieważ na poziomie liści zaznaczyliśmy klucze oraz każdemu wierzchołkowi przydzieliliśmy identyfikatory, dlatego przenosimy się na poziom wyżej. Każdemu wierzchołkowi v z rozważanego poziomu przypisujemy klucz (x,y) , zgodnie z zasadą: $x:=id(low(v))$ oraz $y:=id(high(v))$. Wszystkie klucze porządkujemy leksykograficznie i wpisujemy unikalne klucze do tabelki obok diagramu. Obok klucza dopisujemy kolejny unikalny identyfikator (jeżeli $x=y$ to $id(v):=x$). Tak powstał opisany diagram z rys.3.8c wraz z tabelką obok. Wierzchołki z nowymi identyfikatorami dopisanymi do tabelki przenosimy do diagramu zredukowanego pokazanego na rys.3.8d. Przenieśliśmy zatem wierzchołek z kluczem $(1,2)$ i identyfikatorem 3 oraz kluczem $(2,1)$ z identyfikatorem 4 (na rysunku zaznaczyliśmy te wierzchołki pogrubioną linią).

Przenosimy się na poziom z $index=3$, na którym znajdują się cztery wierzchołki. Możemy każdemu wierzchołkowi nadać klucz, ponieważ każdy wierzchołek z poziomu niższego ($index=4$) ma przypisany unikalny identyfikator (rys.3.8.e). Różne klucze dopisujemy do tabelki w porządku rosnącym. Nadajemy im kolejne identyfikatory. Do diagramu zredukowanego przenosimy trzy wierzchołki z kluczami $(2,3)$, $(4,2)$ oraz $(4,3)$.

Przechodzimy do poziomu $index=2$ i następnie do poziomu $index=1$, w analogiczny sposób generując klucze i przenosząc odpowiednie wierzchołki do zredukowanego diagramu (rys.3.8.g i h).

Zwróćmy uwagę na to, że liczba wierzchołków w zredukowanym diagramie jest równa liczbie unikalnych identyfikatorów z tabelki oraz, co wydaje się oczywiste, każdy wierzchołek w zredukowanym diagramie ma unikalny identyfikator (okaże się to istotne przy omawianiu innych algorytmów).

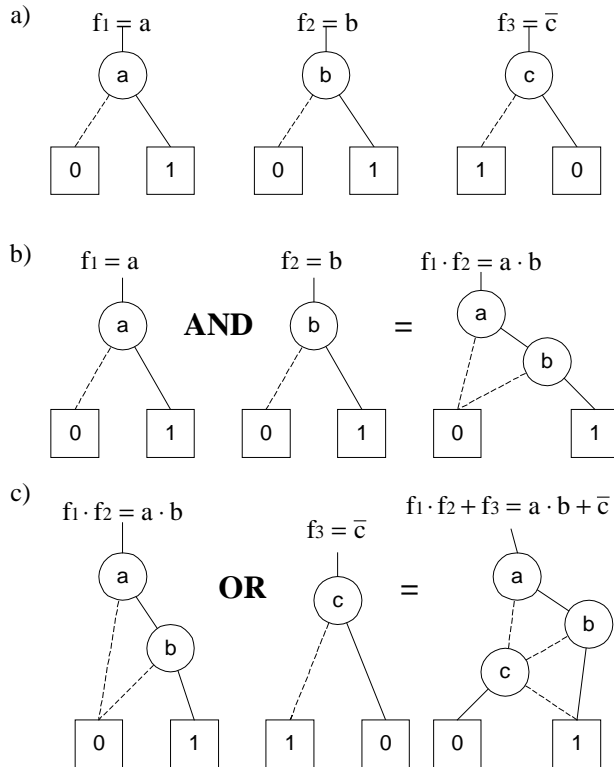
Miejsce na pseudokod REDUCE



Rys. 3.8 Ilustracja algorytmu redukowania binarnego diagramu decyzyjnego

Procedura APPLY

Na początku niniejszego rozdziału powiedzieliśmy, że BDD nie buduje się na podstawie drzewa binarnego danej funkcji. Gdyby tak było, to właściwie reprezentacja funkcji w postaci diagramu niczym by się nie różniła od tablicy prawdy. Najlepiej, gdyby można było budować bardziej złożony diagram na podstawie mniej złożonych („operować na diagramach”).



Rys. 3.9 Ilustracja tworzenia diagramu

Wyjaśnimy ideę tego pomysłu na przykładzie otrzymywania BDD dla dowolnej postaci dysjunkcyjnej, zakładając, że mamy zbudować diagram dla przykładowego wyrażenie $ab + \bar{c}$. Nietrudno zauważyć, że bez problemu potrafimy zbudować diagramy dla każdej zmiennej z osobna (rys.3.9a). Jeżeli mielibyśmy możliwość zbudowania diagramu dla wyrażenia ab poprzez „wymnożenie” diagramów dla zmiennej a i zmiennej b (rys.3.9b) oraz „dodania” wynikowego diagramu do diagramu reprezentującego \bar{c} , to zadanie budowy diagramu byłoby zrealizowane (rys.3.9c).

Procedura APPLY pozwala na wykonywanie dowolnych operacji binarnych na funkcjach przełączających przedstawionych w postaci binarnych diagramów decyzyjnych. Operacja binarna jest reprezentowana przez dowolną funkcję dwóch zmiennych. Zatem mając procedurę APPLY jesteśmy w stanie wykonać między innymi operacje przedstawione na

rys.2.9b i c. Z tego względu znaczenie tej procedury jest bardzo ważne, ponieważ za jej pomocą możemy zbudować diagram dla dowolnej funkcji przełączającej.

Podstawowy pomysł wykorzystany w procedurze APPLY opiera się na rekurencyjnym stosowaniu twierdzenia Shannona:

$$h \text{ op } g = \bar{x}_i(h_{x_i=0} \text{ op } g_{x_i=0}) + x_i(h_{x_i=1} \text{ op } g_{x_i=1})$$

gdzie h oraz g - funkcje przełączające,

op – operator binarny.

Zastosowanie operatora binarnego do funkcji reprezentowanych przez diagramy o korzeniach v_1 i v_2 , czyli $\text{APPLY}(v_1, v_2, \text{op})$, sprowadza się do rozważenia kilku przypadków.

1. Oba wierzchołki v_1 i v_2 są wierzchołkami terminalnymi. W tym przypadku wynikowy diagram zawiera wierzchołek terminalny u z wartością $\text{value}(u) := \text{value}(v_1) \text{ op } \text{value}(v_2)$.
2. Jeśli $\text{index}(v_1) = \text{index}(v_2) = k$, to tworzymy wierzchołek u z $\text{index} = k$ i wywołujemy rekurencyjnie procedurę $\text{APPLY}(\text{low}(v_1), \text{low}(v_2), \text{op})$ aby wygenerować podgraf, którego korzeń stanie się $\text{low}(u)$ oraz $\text{APPLY}(\text{high}(v_1), \text{high}(v_2), \text{op})$ wygeneruje podgraf dla $\text{high}(u)$. Czyli dla tego przypadku $\text{low}(u) = \text{APPLY}(\text{low}(v_1), \text{low}(v_2), \text{op})$ oraz $\text{high}(u) = \text{APPLY}(\text{high}(v_1), \text{high}(v_2), \text{op})$.
3. Jeśli $\text{index}(v_1) = k$ a v_2 jest terminalny bądź $\text{index}(v_2) > k$ (funkcja reprezentowana przez podgraf o korzeniu v_2 jest niezależna od zmiennej x_k), to tworzymy wierzchołek u z $\text{index} = k$ i rekurencyjnie wywołujemy $\text{APPLY}(\text{low}(v_1), v_2, \text{op})$ w celu wygenerowania podgrafu, który stanie się $\text{low}(u)$ a także $\text{APPLY}(\text{high}(v_1), v_2, \text{op})$ aby otrzymać podgraf, który będzie korzeniem $\text{high}(u)$. Czyli, $\text{low}(u) = \text{APPLY}(\text{low}(v_1), v_2, \text{op})$ oraz

$\text{high}(u) = \text{APPLY}(\text{high}(v_1), v_2, \text{op})$. Analogicznie postępujemy, gdy $\text{index}(v_2) = k$ a v_1 jest liściem bądź $\text{index}(v_1) > k$.

Otrzymany diagram należy zminimalizować przy użyciu procedury REDUCE.

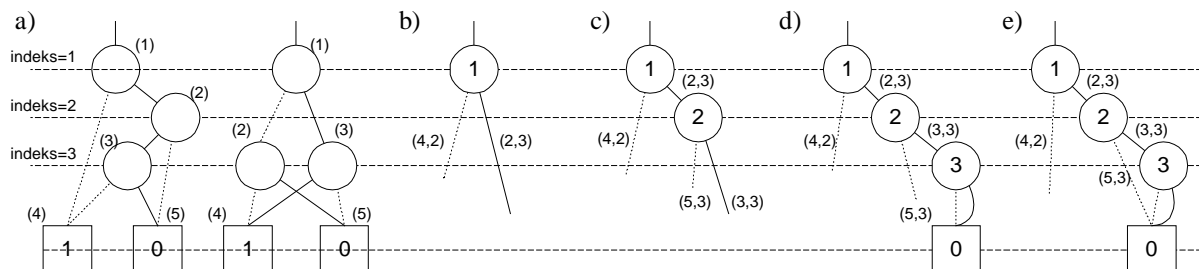
Niestety algorytm ma zbyt dużą czasową złożonością obliczeniową. W szczególnych przypadkach może to być nawet złożoność wykładnicza, ponieważ każde rekurencyjne wywołanie procedury APPLY z przynajmniej jednym wierzchołkiem nieterminalnym, skutkuje kolejnymi dwoma wywołaniami APPLY. Jednak złożoność może być zmniejszona poprzez wprowadzenie do algorytmu dwóch ulepszeń:

- jeśli dla wywołania $\text{APPLY}(v_1, v_2, \text{op})$ otrzymaliśmy już podgraf o korzeniu u , to przy kolejnym rekurencyjnym wywołaniu APPLY z identycznymi argumentami nie musimy ponownie generować korzenia u . Wystarczy zapamiętywać trójki (v_1, v_2, u) i przed wywołaniem APPLY dla kolejnych par wierzchołków sprawdzać czy aby tego wcześniej nie zrobiliśmy. Jeżeli tak, to rezultat wywołania jest zwracany od razu. Omawiane usprawnienie ma decydujący wpływ na zmniejszenie złożoności obliczeniowej.
- jeśli tylko jeden z argumentów wywołania APPLY jest wierzchołkiem terminalnym to wartość tego wierzchołka może być wartością dominującą dla operacji op . Mówimy, że wartość jest dominująca dla operacji op , jeżeli wynik operacji jest znany bez względu na drugi argument. Przykładowo, dla mnożenia logicznego wartością dominującą jest 0, ponieważ $x \cdot 0 = 0$ bez względu na x a dla sumy logicznej ta wartością jest 1. Oczywiście w tym przypadku, możemy zakończyć procedurę zwracając wierzchołek terminalny z *value* określoną przez wartość dominującą. Gdybyśmy tego nie robili, kolejne wywołania generowałyby wierzchołki nadmiarowe.

Przykład:

Niech będą dane dwa diagramy z rys.3.10, na których zilustrujemy działanie procedury APPLY dla operacji mnożenia logicznego (nazwijmy ją AND). Wszystkie wierzchołki oznaczyliśmy unikalnymi identyfikatorami, które wpisaliśmy wewnątrz nawiasów obok każdego z nich.

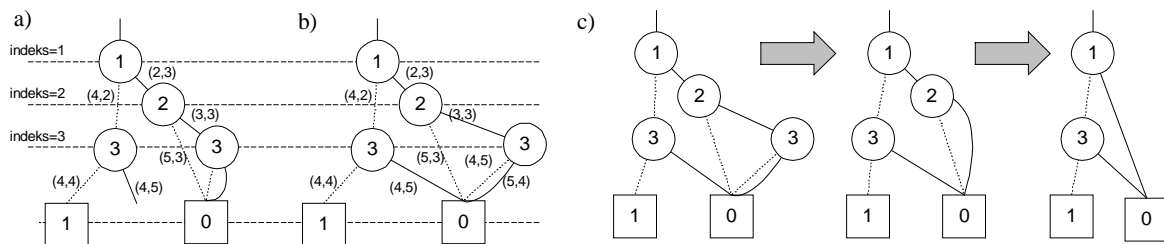
Rozpoczynamy od korzeni obu diagramów. Oba są na jednakowym poziomie, więc powstanie nowy wierzchołek u z tym samym indeksem=1 (przypadek 2). Przy krawędziach wychodzących z tego wierzchołka zaznaczymy z jakich węzłów powstaną węzły potomne. Tak więc $\text{low}(u)$ powstanie z wierzchołka (4) z lewego diagramu i z (2) z prawego (co zapiszemy przy krawędzi przerywanej w postaci (4,2)), natomiast $\text{high}(u)$ otrzymamy z wierzchołków (2) i z (3) odpowiednich diagramów (patrz rys. 3.10b).



Rys. 3.10 Kolejne fazy stosowania procedury APPLY

Przechodzimy na prawo po krawędzi (2,3), gdzie 2 to wierzchołek z pierwszego diagramu, a 3 z drugiego. Nowy wierzchołek u powstanie z węzłów o różnych indeksach (wierzchołek 2 jest na poziomie z $\text{index}=2$ a wierzchołek 3 na poziomie z $\text{index}=3$. Zgodnie z przypadkiem 3 opisanym wyżej, nowy wierzchołek będzie miał mniejszy z indeksów czyli $\text{index}=2$ (będzie

Na koniec wróćmy do krawędzi (4,2). Wierzchołek 4 lewego diagramu jest liściem o wartości 1, a wierzchołek 2 prawego diagramu ma indeks=3 (patrz rys.3.10a). Zachodzi przypadek 3. Ponieważ wierzchołek terminalny ma indeks= $n+1=4$, dlatego nowy wierzchołek będzie miał mniejszy indeks, a jego potomkami są $\text{low}(u)=\text{APPLY}(4,\text{low}(2))=\text{APPLY}(4,4)$ oraz $\text{high}(u)=\text{APPLY}(4,\text{high}(2))=\text{APPLY}(4,5)$. Rozważając $\text{low}(u)$ zauważamy, że wierzchołki 4 w obu diagramach są wierzchołkami terminalnymi o wartości 1 więc wyliczonym wynikiem $\text{APPLY}(4,4)$ będzie węzeł końcowy z wartością 1 (rys.3.11a). Analogicznie dla $\text{high}(u)$ oba wierzchołki też są terminalne, ale o przeciwnych wartościach czyli $1 \text{ AND } 0=0$, co oznacza, że krawędź (4,5) także podłączymy do liścia o wartości 0 (rys.3.11b).



Na koniec diagram z rys.3.11b należy zredukować. Poszczególne fazy procesu redukowania przedstawiliśmy na rys.3.11c.

??

```

procedure APPLY(G1,G2:PVertex; OP);
var Tab2D : tTab2D; // tablica dynamiczna
    function ApplyNodes( v1, v2 : PVertex ):PVertex;
var Ptr_u : PVertex;
    val : Integer;
    vLow1, vLow2, vHigh1, vHigh2 : PVertex;
begin
    Ptr_u := Tab2D[v1.Id-1, v2.Id-1];
    if ( Ptr_u <> nil ) then begin Result := Ptr_u; exit; end;
    val:=ObliczValue(v1.value,v2.value, OP);

```

```

if ( val <> 2 )
then begin
    if val = 0
    then Ptr_u := Terminalny0
    else Ptr_u := Terminalny1;
    Tab2D[v1.Id-1,v2.Id-1]:=Ptr_u;
end
else begin
    // Wierzcholek nieterminalny
    Ptr_u := UtworzNowyWierzcholek;
    Tab2D[v1.Id-1, v2.Id-1] := Ptr_u;
    with Ptr_u^ do begin
        Value := val;
        Index := Min(v1.Index,v2.Index);
        Id := id_globalny;
        Inc(id_globalny);
        if ( v1.Index = Index )
        then begin vLow1 := v1.Low; vHigh1 := v1.High; end
        else begin vLow1 := v1; vHigh1 := v1; end;
        if ( v2.Index = Index )
        then begin vLow2 := v2.Low; vHigh2 := v2.High; end
        else begin vLow2 := v2; vHigh2 := v2; end;
        Low := ApplyNodes( vLow1, vLow2 );
        High := ApplyNodes( vHigh1, vHigh2 );
    end;
end;
Result := Ptr_u;
end;
begin
    UtworzWierzcholkiTerminalne(id_globalny);
    InicjujTablice(Tab2D,G1,G2);
    Root := ApplyNodes(G1,G2);
    ZwolnijTablice(Tab2D);
    Reduce(Root); // redukcja diagramu wynikowego
end;

```

Procedura RESTRICT

Procedura służy do przekształcenia diagramu reprezentującego funkcję $f(x_1, \dots, x_i, \dots, x_n)$ do diagramu funkcji $f(x_1, \dots, a, \dots, x_n)$, gdzie $a \in \{0, 1\}$. Zatem parametrami wywołania procedury jest diagram funkcji, indeks zmiennej i oraz wartość a .

Procedura opiera swoje działanie na przeglądaniu drzewa, dlatego do tego celu może być wykorzystana procedura *PrzeglądajDrzewo* przedstawiona wcześniej. Celem przechodzenia po diagramie jest szukanie wskaźników do wierzchołków v takich, że $index(v)=i$ (wskaźnik może wskazywać korzeń bądź są to wskaźniki do potomków). Gdy taki wierzchołek v zostanie znaleziony zmieniamy wartość wskaźnika tak, by wskazywał teraz $low(v)$, gdy $a=0$ lub wskazywał $high(v)$, gdy $a=1$.

Na koniec wywołujemy procedurę REDUCE.

Przykład:

??

Procedura COMPOSE

Procedura ta służy do obliczenia złożenia funkcji, czyli podstawienia w funkcji h pod jej i -tą zmienną innej funkcji g . Innymi słowy, i -ta zmienna funkcji h będzie zawsze przyjmować wartości funkcji g . Złożenie funkcji możemy także wyrazić wprost z twierdzenia Shannona:

$$h_{x_i=g} = g \cdot h_{x_i=1} + \bar{g} \cdot h_{x_i=0}$$

Wynika z powyższego rozwinięcia, że do procedury COMPOSE wystarcza wykorzystanie RESTRICT i zmodyfikowanej procedury APPLY. Zwróćmy uwagę, że powyższe rozwinięcie da się zapisać za pomocą trójelementowego operatora ITE:

$$ITE(a,b,c) = ab + \bar{a}c$$

Przykład:

??

Jest jaki jest ale jest!!!! SORRY

Literatura

- [1] Harrison M.A., Wstęp do teorii sieci przełączających i teorii automatów, PWN, Warszawa 1973,
[2] Bryant R.E., Graph-Based Algorithms for Boolean Function Manipulation, IEEE Trans. on Computers, vol.C-35, no.8, August 1986, str. 677-691,

ĆWICZENIA

- Znajdź wartość poniższych funkcji dla $a=0$, $b=1$, $c=0$, $d=1$:
 - $f(abcd) = \overline{b}d + \overline{a}b\overline{c}d$
 - $f(abcd) = \overline{b}d + \overline{a}b(a + \overline{b} + \overline{c}d)$
 - $f(abcd) = (a + \overline{b})(c + \overline{d})(ab + c)$
 - $f(abcd) = c\overline{d}(\overline{d} + \overline{a}(b + \overline{c}d))$
- Podaj tablice prawdy dla poniższych funkcji:
 - $f(abc) = \overline{a}\overline{b} + bc + \overline{a} \cdot \overline{c}$
 - $f(abc) = ac + \overline{a}b + \overline{b}\overline{c}$
 - $f(abc) = \overline{abc} + \overline{abc}$
- Podaj kanoniczne postacie dysjunkcyjne dla funkcji z zadania 1 oraz 2.
- Czy poniższe równości są prawdziwe?
 - $ac + \overline{a}b + b\overline{c} = \overline{abc} \cdot \overline{\overline{abc}}$
 - $\overline{a}\overline{b} + bc + \overline{a} \cdot \overline{c} = (\overline{a} + \overline{b} + c)(a + b + \overline{c})$
 - $(a + \overline{b} + c)(a + \overline{b} + \overline{c}) = \overline{b} + a\overline{c} + ac$
- Udowodnić, że dla dowolnej funkcji zachodzi:
 - $x_1 \cdot f(x_1, x_2, \dots, x_n) = x_1 \cdot f(1, x_2, \dots, x_n)$,
 - $\overline{x}_1 \cdot f(x_1, x_2, \dots, x_n) = \overline{x}_1 \cdot f(0, x_2, \dots, x_n)$
- Podać kanoniczną postać dysjunkcyjną dla funkcji $f(x_1x_2)=1$.
- Udowodnić twierdzenie Shannona o rozwinięciu funkcji przełączającej, wykorzystując w tym celu tablicę prawdy. (Czy zadanie jest sensownie sformułowane?).
- Pokaż, że istnieje 2^{2^n} funkcji n zmiennych a implikantów prostych **może być** $3^n/n$.
- Podaj wszystkie implikanty i implikanty proste funkcji $f(x_1x_2x_3) = \sum(0,2,3,4,5,7)$.
- Podaj wszystkie implikanty i implikanty proste funkcji $f(abc) = \sum(1,6)$.
- Podaj tablice prawdy dla następujących funkcji:
 - $f(abcd) = ab + \overline{b}c + \overline{c}d + \overline{d}a$
 - $f(abcd) = \overline{\overline{(a+b+\overline{c})}} + d$
 - $f = \overline{a \cdot x \cdot \overline{y} + \overline{z}}$
- Czy nie w pełni określonych funkcji n zmiennych jest $3^A - 1$, przy $A = 2^n$?
- Pokaż, że liczba funkcji przełączających zależna dokładnie od n zmiennych może być określona następującym wzorem rekurencyjnym:

$$A_n = 2^{2^n} - \sum_{i=0}^{n-1} \binom{n}{i} \cdot A_i, \quad A_0 = 2,$$

gdzie A_i - liczba funkcji przełączających zależna dokładnie od i zmiennych. Funkcja zależna dokładnie od n zmiennych to taka, dla której nie istnieje zmienna, spełniająca równanie $f(x_1, x_2, \dots, x_{k-1}, 0, \dots) = f(x_1, x_2, \dots, x_{k-1}, 1, \dots)$.

4 UKŁADY KOMBINACYJNE

Kombinacyjnym układem nazywamy układ logiczny, w którym wartość wyjść zależy tylko i wyłącznie od wartości, jakie przyjmują zmienne wejściowe w danej chwili. Stąd też ich inna nazwa układ bez pamięci. Układ kombinacyjny jest praktyczną realizacją jednej lub wielu funkcji przełączających.

4.1 Wprowadzenie

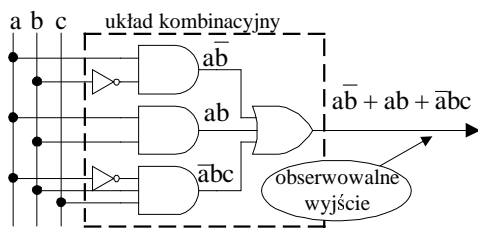
Z definicji funkcji przełączającej wynika, że dowolną funkcję możemy określić wykorzystując trzy podstawowe operacje - dodawanie, mnożenie oraz dopełnienie. Można się więc spodziewać, że układ kombinacyjny realizujący daną funkcję będzie zawierał operatory realizujące trzy wymienione operacje algebry Boole'a, połączone w odpowiedni sposób. Operatory te nazwiemy typowymi elementami logicznymi. Przyjęła się także zwyczajowa nazwa - bramka logiczna (*logical gate*), lub po prostu bramka. Oznaczenia bramek oraz ich angielskie nazwy przedstawiono na rys. 4.1. Nie są one zgodne z polską normą branżową BN-71/3100-01, ale są najbardziej rozpowszechnione w literaturze przedmiotu [1][2]. Zupełnie inny sposób rysowania bramek i układów logicznych zalecany jest w normie ANSI/IEEE Std 91-1984, który obowiązuje we wszystkich schematach układów tworzonych dla departamentu obrony USA [3].

Nazwa bramki	AND	OR	NOT
Realizowana funkcja	$a \cdot b$	$a + b$	\bar{a}
Oznaczenia używane w podręczniku			
Zalecenia polskie i angielskie			

Rys. 4.1 Oznaczenia podstawowych bramek logicznych

Realizację układową dla zadanej funkcji $f(abc) = \bar{a}\bar{b} + ab + \bar{a}bc$, pokazaliśmy na rys. 4.2.

W realizacjach układów kombinacyjnych, przede wszystkim na początku ich rozwoju, bardzo ważne było następujące zagadnienie. Dana jest funkcja przełączająca f w postaci sumy iloczynów. Należy znaleźć takie wyrażenie boolowskie opisujące funkcję f , by wyrażenie to było nadal sumą iloczynów i minimalizowało zadane kryterium kosztu realizacji układowej. Przez koszt rozumiano liczbę wejść bramek oraz liczbę elementów wykorzystanych do realizacji układowej zadanej funkcji. Zauważmy, że powyższą funkcję przełączającą możemy w bardzo prosty sposób przekształcić do postaci:

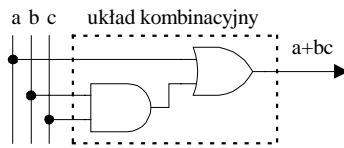


Rys. 4.2 Przykładowy układ kombinacyjny

$$f(abc) = \bar{a}\bar{b} + ab + \bar{a}bc = a(\bar{b} + b) + \bar{a}bc = a + \bar{a}bc = a + bc$$

Realizację układową tej postaci funkcji przedstawiono na rys. 4.3. Pozostawiamy Czytelnikowi do wykazania, dlaczego $x + \bar{x}y = x + y$ (własność tę wykorzystaliśmy w powyższym przekształceniu).

Z dotychczasowych rozważań wynika, że drogą przekształceń formalnych możemy doprowadzić do „oszczędniejszej” realizacji układowej zadanej funkcji.



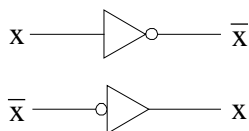
Rys. 4.3 Oszczędna realizacja funkcji przełączającej

Oczywiście sposób ten jest przykładem „metody akademickiej” i nie ma większego zastosowania w praktyce, gdzie zazwyczaj występują funkcje przełączające dużej liczby zmiennych.

Zagadnienia związane z otrzymaniem oszczędnych układów kombinacyjnych nazywa się zagadnieniami minimalizacji funkcji przełączających. Wrócimy do nich w dalszej części podręcznika.

4.2 Wartości logiczne i sygnały elektryczne

Pojedynczą bramkę można traktować jako elementarny układ realizujący odpowiednią funkcję przełączającą, opisaną np. za pomocą tablicy prawdy. W tablicy tej występują wartości logiczne 0 oraz 1, które w realizacji układowej mogą być reprezentowane przez dyskretne sygnały elektryczne. Sygnały opisywane są za pomocą wielu parametrów, między innymi napięcia. Na razie założymy, że sygnał dyskretny w ustalonych chwilach czasowych może mieć poziom niskiego albo wysokiego napięcia. Projektant układu może przyjąć, że poziomy te reprezentować będą, odpowiednio, wartości logiczne 0 albo 1. Ma on do wyboru dwa sposoby przyporządkowania: po pierwsze, wysoki poziom napięcia reprezentuje wartość logiczną 1, niski - wartość 0 (logika dodatnia); po drugie, niski



Rys. 4.4 Nazwy sygnałów

poziom napięcia reprezentuje wartość logiczną 1, wysoki - wartość 0 (logika ujemna). O sygnale, który odpowiada wartości logicznej 1(0), mówimy, że jest ustawiony (nieustawiony) (*asserted, deasserted*) bądź aktywny (nieaktywny) (*active, negated*). W logice dodatniej (ujemnej) sygnał jest ustawiony, gdy odpowiada mu wysoki (niski) poziom napięcia.

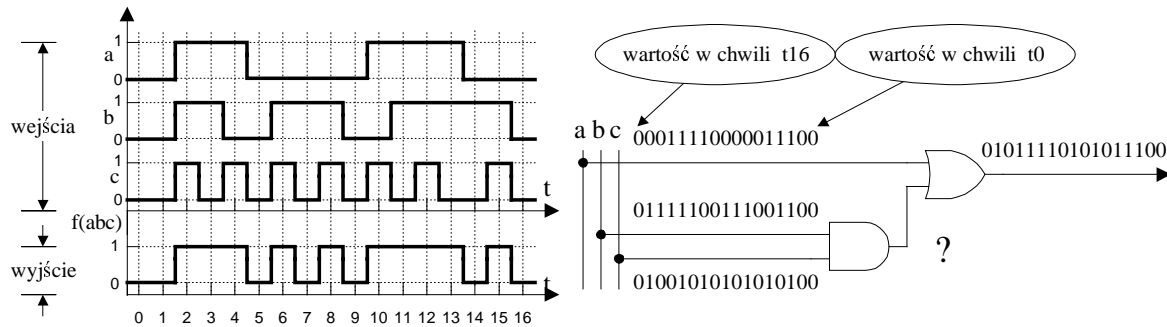
Przyjmuje się jako zasadę, że nazwa sygnału reprezentującego zmienną logiczną powinna wskazywać na jego charakter [3]. W logice dodatniej nazwa sygnału odpowiada dokładnie nazwie zmiennej (w logice ujemnej nazwa sygnału będzie dopełniona), dlatego w podręczniku przede wszystkim będziemy wykorzystywać logikę dodatnią. Ilustracją przyjętej zasady w stosunku do bramki NOT jest rys. 4.4.

4.3 Analiza układów kombinacyjnych

Układ kombinacyjny projektuje się przekształcając słowny opis zachowania się układu do postaci wyrażeń boolowskich (funkcji), które z kolei realizuje się z wykorzystaniem bramek. Analiza układów kombinacyjnych jest procesem odwrotnym. Mając dany układ dokonujemy analizy, aby określić czy jego zachowanie jest zgodne ze specyfikacją, albo czy możliwe jest jego przekształcenie do postaci np. zawierającej tylko bramki dwuwejściowe. Zazwyczaj wynikiem analizy są wyrażenia boolowskie (porównaj np. rys. 4.2), tablice prawdy bądź przebiegi czasowe (diagramy czasowe).

Przebiegi czasowe są graficzną reprezentacją zależności między wejściowymi i wyjściowymi sygnałami występującymi w układzie przełączającym, dlatego mogą być wyświetlane np. na ekranie oscyloskopu czy analizatora logicznego (mogą być też wynikiem działania programu symulacyjnego). Oczywiście, na przebiegu czasowym mogą być pokazane także sygnały występujące wewnątrz układu. Na rysunku 4.5 pokazaliśmy przykładowe przebiegi czasowe dla układu z rys. 4.3 (sekwencje wartości zmiennych wejściowych są dobrane przypadkowo, czego nie można powiedzieć o sekwencji wyjściowej). Zwróćmy szczególną uwagę na oś czasu, którą podzieliliśmy na równej długości odcinki. W każdej chwili czasowej sygnały wejściowe i wyjściowe przyjmują ustalone wartości. Dla przykładu w chwili t_3

$a=b=c=1$ a w chwili t_4 $a=b=1$, $c=0$. Na schemacie układu, nad każdym przewodem sygnału wejściowego oraz wyjściowego nanieśliśmy wartości sygnałów poczynając od chwili t_0 do t_{16} .



Rys. 4.5 Przykładowe przebiegi czasowe

Rzeczywiste przebiegi czasowe nie są takie „idealne” jak na rys. 4.5. Określenie „idealne” przede wszystkim dotyczy założenia, że sygnał zmienia swoją wartość w nieskończenie krótkim czasie oraz reakcja bramki na taką zmianę jest natychmiastowa. Założenia te są słuszne w algebrze Boole’a, ponieważ nie występuje w niej obiekt reprezentujący czas, lecz nie odpowiadają zachowaniom rzeczywistych realizacji układowych, w których zmiana wartości sygnału trwa ustalony czas oraz bramki charakteryzują się pewną bezwładnością, reagując dopiero po pewnym czasie na zmianę wartości sygnału wejściowego.

Wszystkie parametry czasowe podzespołów cyfrowych są podawane w katalogach firm je produkujących. Wielkość tych czasów zależy od złożoności podzespołu, a także od technologii wykonania. Czasy opóźnień są rzędu kilku nanosekund (dla porównania, w próżni światło przebiega ok. 30 centymetrów w czasie jednej nanosekundy, czyli 10^{-9} sekundy).

Szybkość działania układu zależy nie tylko od parametrów czasowych poszczególnych bramek, ale także od sposobu ich połączenia. Szybsze są te układy, które mają mniejszą liczbę bramek na drodze od wejścia do wyjścia. Dlatego na etapie konstruowania układu należy zdecydować o strukturze (topologii) układu. W podręczniku będziemy się interesować przede wszystkim dwupoziomowymi układami logicznymi.

4.4 Synteza układu kombinacyjnego

Celem syntezy jest przekształcenie opisu, wyrażonego w postaci wyrażenia boolowskiego, tablicy prawdy bądź innego formalizmu, w układ kombinacyjny zachowujący się zgodnie z tym opisem. Zazwyczaj zależy nam, by zsyntezowany układ spełniał przyjęte wcześniej kryterium kosztu lub inne dodatkowe wymagania związane np. z szybkością działania, łatwością wykrywania uszkodzeń, itp.

W dalszej części podręcznika przedstawiliśmy wiele zagadnień związanych z syntezą układów logicznych. Konstruktor układów powinien uwzględnić wiele fizycznych cech układu przełączającego, w szczególności: czasy propagacji sygnałów, ograniczenia związane z obciążalnością wejść i wyjść, zużyciem energii czy rozmiarami układu. Niektórymi tylko cechami zajmiemy się w odpowiednich rozdziałach niniejszego podręcznika.

4.5 Wybrane zagadnienia praktyczne

Dotychczas zajmowaliśmy się przede wszystkim aspektami matematycznymi reprezentowania układów kombinacyjnych (algebra Boole’a, wyrażenie i funkcja boolowska). W tym rozdziale trochę więcej uwagi poświęcimy aspektom praktycznym. Należy o nich pamiętać, gdy zaczniemy realizować dowolny układ logiczny. Oczywiście w rozdziale tym nie wyczerpiemy wszystkich zagadnień - wykracza to poza ramy tego podręcznika.

Układ logiczny możemy zaprojektować i zrealizować na bardzo wiele sposobów, w zależności od tego jakich użyjemy elementów - przekąznika, lampy elektronowej (to już historia) czy półprzewodnikowego tranzystora. Umiejętność upakowania bardzo wielu tranzystorów na płatku krzemu oraz ich odpowiedniego połączenia dała początek układom scalonym.

Z układami scalonymi nierozzerwalnie jest związane pojęcie technologii wykonania tranzystora. Najwcześniejszą jest technologia bipolarna, której przedstawicielem jest rodzina TTL (*transistor-transistor logic*), wprowadzona już w roku 1960. Inną bardzo popularną technologią jest CMOS (*complementary metal-oxide semiconductor*). Elementy logiczne zrealizowane w tej samej technologii mogą być łączone ze sobą, aby realizować zadaną funkcję przełączającą. Zazwyczaj nie łączy się elementów logicznych wykonanych w różnych technologiach, chociaż bardzo duża popularność rodziny TTL spowodowała, że istnieją wykonania CMOS zgodne z rodziną TTL (czyli te elementy możemy ze sobą łączyć)[3].

Tabela 4.1

Cecha	TTL	CMOS
Opóźnienie bramek	małe	średnie
Stopień upakowania	mały	wysoki
Pobierana moc	duża	mała
Koszt wykonania	mały	średni
Obciążalność	dostateczna	dobra
Odporność na zakłócenia	dobra	dobra

Tabela 4.1 przedstawia podstawowe różnice jakościowe występujące między technologiami TTL i CMOS.

Opóźnienie bramki: Jeżeli zmiana na wejściu bramki powoduje zmianę wartości na wyjściu, to przez czas opóźnienia bramki będziemy rozumieć różnicę czasu między tymi zmianami. Zazwyczaj opóźnienie rośnie, gdy zwiększamy liczbę wejść. Opóźnienie bramek bipolarnych jest małe, dlatego są one szybsze niż bramki CMOS.

Stopień upakowania: Realizując zadaną funkcję przełączającą w wybranej technologii zużywamy pewną powierzchnię np. na płatku krzemu. Im więcej tranzystorów upakujemy na mniejszej powierzchni tym większy stopień upakowania. Z tabeli widać, że technologia CMOS pozwala nam zrealizować bardziej złożone funkcję (np. pod względem wymaganej liczby bramek) na zadanej powierzchni.

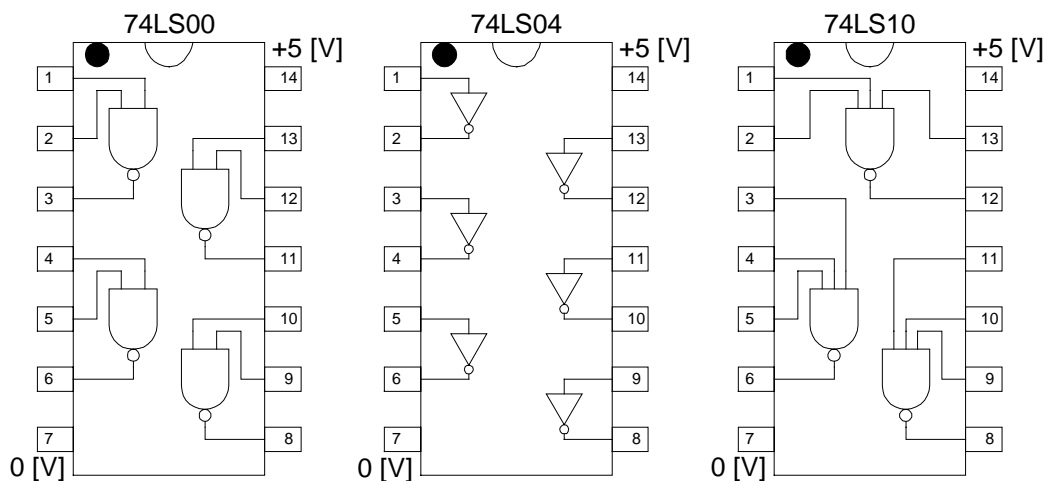
Pobierana i rozpraszana moc: Bramki pobierają ze źródła zasilania pewną moc na realizację swoich funkcji. Przede wszystkim moc potrzebna jest na przełączanie się między różnymi wartościami logicznymi (im większa szybkość przełączania tym większa jest potrzebna moc). Wytwarzają przy tym pewną ilość ciepła, która musi być odprowadzone z układu. Bramki bipolarne pobierają większą moc oraz wytwarzają większą ilość ciepła. Bramki CMOS zyskały swoją popularność głównie dzięki niewielkiemu zapotrzebowaniu na moc, co szczególnie jest istotne, gdy układy zasilane są z baterii (np. w przenośnych komputerach).

Obciążalność wyjść: Przez obciążalność (*fan-out*) rozumiemy maksymalną liczbę wejść bramek, które mogą być bezpośrednio połączone z wyjściem pewnej bramki bez naruszenia normalnych warunków jej pracy (należy się domyślać, że jeżeli zostaną one naruszone, to bramka przestaje działać prawidłowo). Można zatem powiedzieć, że jest to cecha bezpośrednio związana z możliwością projektowania mniej lub bardziej złożonych układów. Szybkość działania bramek bipolarnych (praktycznie) nie ulega zmianie przy większym obciążeniu, ale jakość sygnału znacznie się obniża. Wartość obciążenia podawana jest zwykle w katalogach.

Odporność na zakłócenia: Wartość logiczna reprezentowana jest przez ustalony poziom napięcia. Jeżeli poziom napięcia ulegnie zwiększeniu (zmniejszeniu) o pewną wielkość a bramki nadal interpretują nowe napięcie jak poprawną wartość logiczną, to mówimy o odporności na zakłócenia. Oczywiście im większe odchylenie napięcia, tym większa odporność na zakłócenia.

Elementy elektroniczne napylone na płatku krzemu (*die*) podlegają procesowi opakowania, czyli uzyskania postaci w jakiej użytkownik dostaje go w sklepie. Obudowa jest plastikowa albo ceramiczna. Układy poklasyfikowano w zależności od liczby bramek przypadających na

jedno opakowanie i tak na przykład układy zawierające od 1 do 20 bramek nazwano SSI (*small-scale integration*). Większość układów SSI udostępniana jest w postaci dwurzędowej obudowy, po 7 wyprowadzeń z każdej strony (*14-pin dual in-line pin, DIP*). Każde wyprowadzenie jest ponumerowane, a jego znaczenie jest podane w katalogu firmowym (zazwyczaj są to wejścia i wyjścia bramek, a także masa i zasilanie). Ponieważ DIP może zawierać różne bramki, dlatego za przykładem firmy Texas Instruments zaczęto każde opakowanie znakować numerem wskazującym na jego zawartość. Oznaczenie to rozpoczyna się cyframi 74 (zastosowania militarne rozpoczynają się od 54) a dalsza część wskazuje na zawartość opakowania. Na przykład 74x30 oznacza bramkę NAND (o której oczywiście powiemy w dalszej części podręcznika). Tajemniczy x w oznaczeniu wskazuje na technologie wykonania tej bramki. W miejsce x pojawiają się takie oznaczenia jak HC, HCT, AC, ACT itp., których znaczenia nie będziemy tutaj omawiać. Najważniejsze, że 74HC30, 74ACT30 czy 74AC30 oznacza bramkę NAND. Wybrane przykłady układów TTL w obudowach dwurzędowych przedstawiliśmy na rys. 4.6. Zwróćmy uwagę na to, że wyprowadzenia 7 i 14 doprowadzają zasilanie do układu scalonego (choć są wyjątki od tej zasady np. 74LS76). Numeracja



Rys. 4.6 Przykłady trzech układów scalonych TTL w obudowach dwurzędowych

wyprowadzeń rozpoczyna się od wyprowadzenia, przy którym jest wgłębienie (numer 1) i przebiega w kierunku przeciwnym do wskazówek zegara po kolejnych wyprowadzeniach, kończąc się na numerze 14.

Opakowania zawierające układy, które w przeliczeniu na liczbę bramek musiałyby ich zawierać od 20 do ok. 200 nazywamy MSI (*medium-scale integration*). Typowe MSI zawierają bloki funkcjonalne do których zaliczamy rejestry, liczniki, sumatory itd. Oczywiście opakowanie MSI, w zależności jaki blok funkcjonalny zawiera, może mieć większą liczbę wyprowadzeń (zazwyczaj 20, 28 a nawet 64).

Opakowania LSI (*large-scale integration*) zawierają układy, które mają od 200 do 200000 równoważnych (przeliczeniowych) bramek. Do nich zaliczamy pamięci, mikroprocesory czy programowalne urządzenia logiczne.

Podawanie górnych wartości liczby równoważnych bramek przypadających na kolejny stopień scalenia jest dość anachroniczne, dlatego powiemy tutaj, że VLSI (*very large-scale integration*) to miliony tranzystorów w jednym opakowaniu (np. 3 miliony tranzystorów przypada na mikroprocesor Pentium).

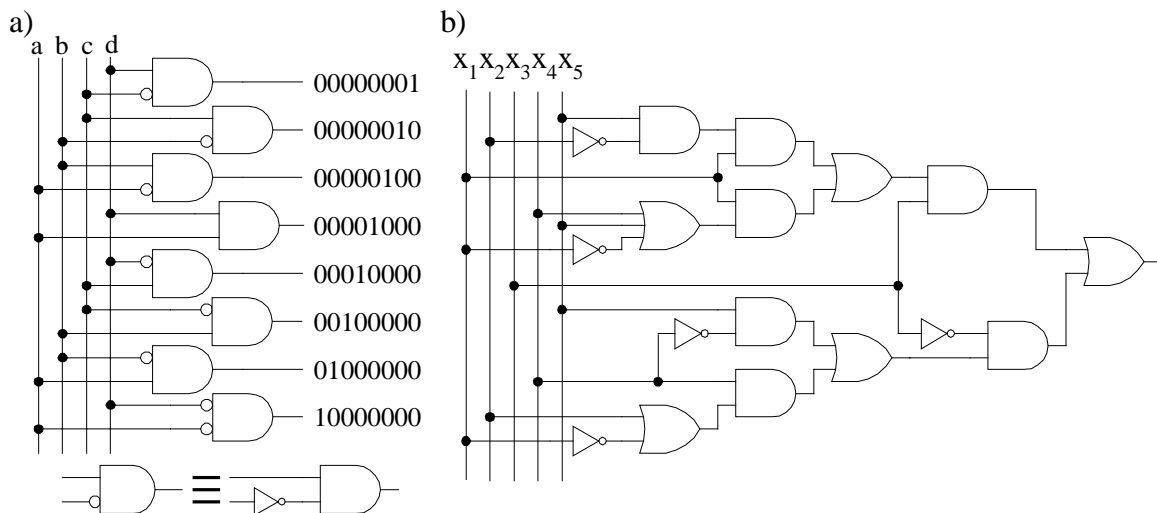
Raczej nie spotyka się nazw dla określenia kolejnego (?) stopnia scalenia, dlatego proponuję Czytelnikowi wymyślenie własnej nazwy (zakładając oczywiście nagłą potrzebę jej wprowadzenia).

Literatura

- [1] Hurst S.L., The logical processing of digital systems, Crane Russak, Inc., New York, 1979
- [2] Pieńkos J., Turczyński J., Układy scalone TTL w systemach cyfrowych, WKiŁ, 1986
- [3] Wakerly J.F., Digital Design Principles and Practices, 2nd Edition, Prentice Hall, 1994

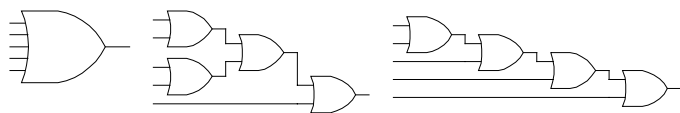
ĆWICZENIA

1. Podaj realizacje układowe następujących funkcji przełączających:
 - a) $f(abc) = \Sigma(0,1,4,5)$,
 - b) $f(abc) = \Sigma(2,3,6,7)$,
 - c) $f(abc) = \Sigma(0,1,3,4,6,7)$
 - d) $f(abcd) = \Sigma(0,1,2,3,8,9,10,11,13,15)$,
 - e) $f(abc) = 1$
2. Uprościć układy otrzymane w zadaniu 1 wykorzystując przekształcenia formalne. Układ uproszczony jest lepszy, gdy zawiera mniej bramek oraz bramki zawierają mniejszą liczbę wejść.
3. Na rys. 4.5 w miejsce znaku zapytania wpisz sekwencję zer i jedynek.
4. Jakie wartości powinny przyjmować zmienne a,b,c oraz d, by na wyjściach układu z rys. 4.6a otrzymać przedstawione tam wartości.



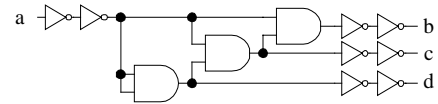
Rys. 4.7 Przykładowe układy kombinacyjne

5. W układzie z rys. 4.7b należy nadać stałe wartości zmiennym wejściowym x_i ($1 \leq i \neq j \leq 5$) tak, by zmieniając wartość zmiennej x_j uzyskać zmiany wartości sygnału na wyjściu układu. Rozpocznij od $j=3$.
6. Przyjmijmy, że **and** i **or** są funkcjami przełączającymi, realizowanymi odpowiednio przez bramki AND i OR. Czy prawdą jest, że w logice ujemnej rola tych bramek ulega „odwróceniu”?
7. Układy z rys. 4.8 realizują tę samą funkcję. Przeanalizuj te układy z punktu widzenia szybkości działania.



Rys. 4.8 Układy wielopoziomowe

8. Przyjmijmy, że układ jest bezpętlowy wtedy i tylko wtedy, gdy startując z dowolnego punktu w układzie i przechodząc, zgodnie z połączeniami, przez bramki od wejść do wyjść, niemożliwe jest dwukrotne osiągnięcie tego samego punktu. Czy układ kombinacyjny jest układem bezpętlowym?
9. Pętlą sprzężenia zwrotnego nazwiemy połączenie wyjścia pewnej bramki z wejściem innej bramki, uczestniczącej w generowaniu sygnału wejściowego tej pierwszej. Czy układ kombinacyjny ma pętlę sprzężenia zwrotnego? Czy układ bezpętlowy (zadanie 8) jest równoważny układowi bez pętli sprzężenia zwrotnego?
10. Przeanalizujmy układ z rys. 4.9 pod względem opóźnień. Przyjmij opóźnienia każdej z bramek. Narysuj przebieg czasowy dla wyjść układu, gdy wartość zmiennej wejściowej a wielokrotnie zmienimy.



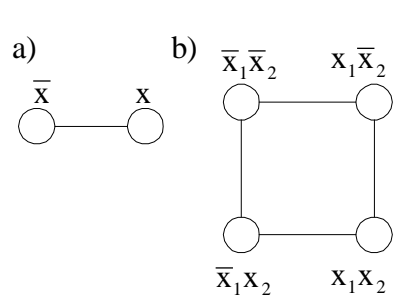
Rys. 4.9 Układ do analizy

5 GEOMETRYCZNA REPREZENTACJA FUNKCJI BOOŁOWSKIEJ

5.1 Przestrzenie n-wymiarowe

Na wykładach z analizy matematycznej dość dużo uwagi poświęca się funkcjom zależnym od jednej zmiennej, których reprezentacją graficzną jest wykres narysowany na płaszczyźnie. Wykres jest przydatny do przedstawienia wybranych własności funkcji. Rysowanie wykresów funkcji zależnych od większej liczby zmiennych natrafia na znane kłopoty, związane z wyobrażeniem sobie przestrzeni cztero- i więcej wymiarowych.

Funkcje boolowskie są funkcjami przyjmującymi wartości ze zbioru $\{0,1\}$ i zależnymi od wielu zmiennych. Dlatego należy przypuszczać, że „wykres” funkcji boolowskiej będzie miał postać odmienną od wykresu funkcji liczb rzeczywistych. Będzie to pewna geometryczna reprezentacja funkcji, która zostanie zanurzona w odpowiednio skonstruowanej przestrzeni wielowymiarowej. Na początek rozważmy reprezentację geometryczną funkcji jednej zmiennej. Przyjmijmy, że przez pojęcie przestrzeni jednowymiarowej będziemy rozumieć obraz graficzny przedstawiony na rys. 5.1a.



Rys. 5.1 Przestrzeń: a - jednowymiarowa, b - dwuwymiarowa

Dla przypadku funkcji dwóch zmiennych liczba możliwych iloczynów zupełnych wynosi 4, a zatem w przestrzeni dwuwymiarowej występują 4 wierzchołki (rys. 5.1b). Każdy wierzchołek opisany jest iloczynem zupełnym. Pogrubiając odpowiedni wierzchołek przestrzeni, będziemy zaznaczać fakt wystąpienia w kanonicznej postaci dysjunkcyjnej iloczynu zupełnego opisującego rozważany wierzchołek. Bezpośrednie przedstawienie graficzne przestrzeni o wymiarze większym niż dwa napotyka już na pewne trudności pojęciowe. Dlatego niżej przytaczamy algorytm, umożliwiający formalną budowę takich przestrzeni o dowolnej liczbie wymiarów.

Algorytm

1. Rysowanie przestrzeni n-wymiarowej zaczynamy, mając daną przestrzeń (n-1)-wymiarową.
2. Ze wszystkich wierzchołków przestrzeni (n-1)-wymiarowej wykreślamy w dowolnym kierunku równoległe odcinki, których końce łączymy ze sobą.
3. Wierzchołki przestrzeni n-wymiarowej opisujemy następująco:
 - a) przepisujemy wartości punktów przestrzeni (n-1) wymiarowej wzdłuż poprzednio wykreślonych odcinków,
 - b) dopisujemy do najmłodszej pozycji nowo utworzonych wierzchołków wartość x_n ,
 - c) do wierzchołków poprzednio istniejących dopisujemy na najmłodszej pozycji \bar{x}_n .

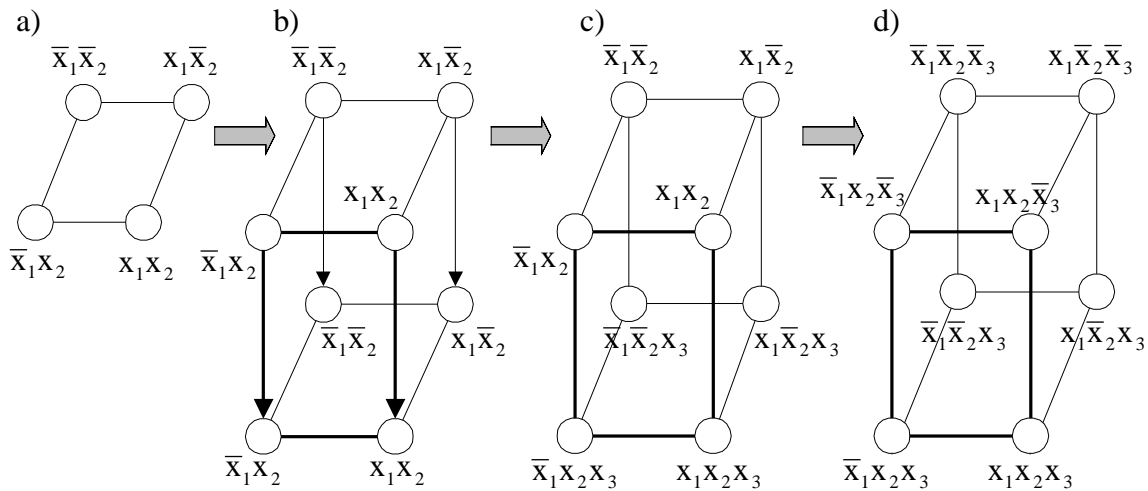
Przykład:

Zilustrujemy powyższy algorytm przedstawiając kolejne fazy powstawania przestrzeni trójwymiarowej. Oczywiście za podstawę przyjmiemy przestrzeń dwuwymiarową (rys. 5.2a). Dysponując przestrzenią trójwymiarową, możemy narysować przestrzeń czterowymiarową itd. W pracy [1] podano przykład przestrzeni siedmiowymiarowej.

Wierzchołki przestrzeni mogą być oznaczane inaczej, na przykład tak, jak to przedstawiono na rys. 5.3. Dokładnie analizując algorytm oraz przyglądając się przestrzeni trójwymiarowej (nie tylko), dochodzimy do wniosku:

Wniosek 5.1

Oznaczenia wierzchołków połączonych pojedynczym odcinkiem różnią się na jednej pozycji. Wierzchołki te nazywamy śsąsiednimi (*adjacent*).



Rys. 5.2 Tworzenie przestrzeni trójwymiarowej

Wniosek 5.2

Przestrzeń n-wymiarowa stanowi odpowiednio połączone wierzchołki reprezentujące wszystkie możliwe iloczyny zupełne.

5.2 Reprezentacja funkcji boolowskiej w przestrzeni n-wymiarowej

Jeżeli pogrubione wierzchołki przestrzeni n-wymiarowej oznaczać będą $f(x_1, x_2, \dots, x_n) = 1$, to przykładowa funkcja $f(x_1, x_2, \dots, x_n) = \bar{x}_1 \bar{x}_2 \bar{x}_3 + x_1 \bar{x}_2 x_3 + \bar{x}_1 x_2 x_3 + x_1 x_2 \bar{x}_3$ może być reprezentowana tak, jak to pokazaliśmy na rys. 5.4a (oznaczenia wierzchołków pozostały identyczne jak na rys. 5.2d).

Rozważmy teraz kilka przykładów funkcji przełączających oraz ich reprezentacji graficznych. Weźmy dla przykładu funkcje z rys. 5.4c. Z reprezentacji graficznej wynika, że

$$f(x_1, x_2, x_3) = \bar{x}_1 x_2 \bar{x}_3 + x_1 x_2 \bar{x}_3 + x_1 \bar{x}_2 \bar{x}_3 + x_1 x_2 \bar{x}_3 = (\bar{x}_1 + x_1) x_2 \bar{x}_3 + (\bar{x}_2 + x_2) x_1 \bar{x}_3 = x_2 \bar{x}_3 + x_1 \bar{x}_3$$

Biorąc pod uwagę rysunek oraz powyższe przekształcenie, możemy wyciągnąć następujący wniosek:

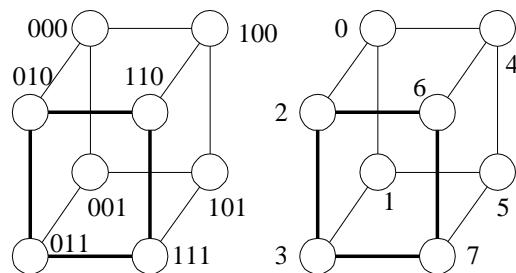
Wniosek 5.3

Połączenie dwóch sąsiednich wierzchołków przestrzeni, dla których $f(X)=1$, upraszcza nam zapis funkcji przełączającej, tzn. zamiast sumy dwóch iloczynów zupełnych otrzymujemy jeden iloczyn zawierający (n-1) literałów.

Uwzględniając przykład funkcji z rys. 5.4e, możemy nasze spostrzeżenia uogólnić, stwierdzając:

Wniosek 5.4

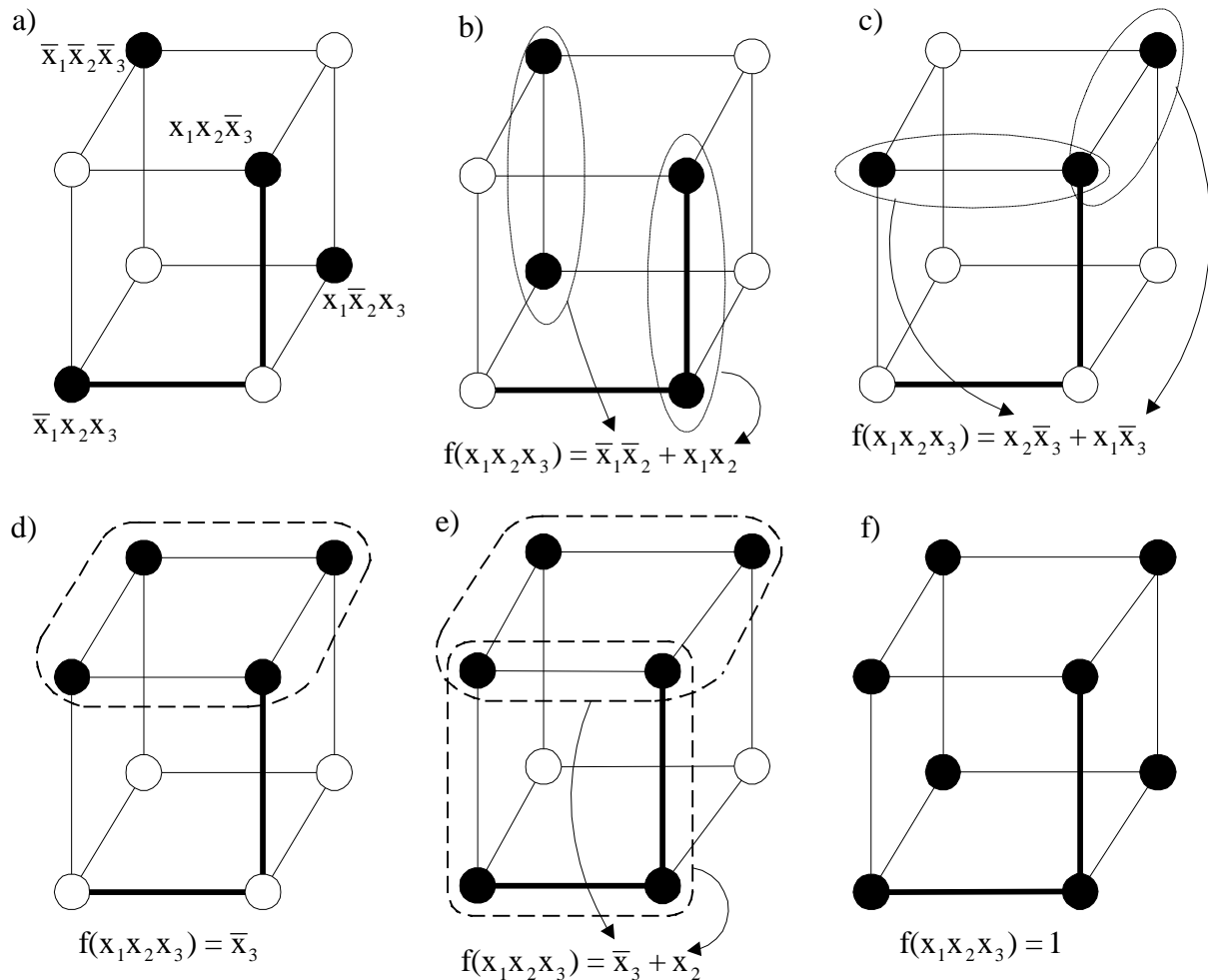
Grupie 2^A iloczynów zupełnych, z których każdy jest sąsiedni w stosunku do A innych iloczynów zupełnych tej grupy, można przyporządkować iloczyn zawierający (n-A) literałów, gdzie n - liczba zmiennych funkcji (zauważmy, że A jest wykładnikiem dwójki).



Rys. 5.3 Oznaczenia wierzchołków

Innymi słowy, na podstawie reprezentacji graficznej danej funkcji przełączającej możemy dokonać jej minimalizacji, łącząc sąsiednie iloczyny zupełne w grupy zawierające 2^A iloczynów.

Oczywiście, będziemy się starać połączyć największą liczbę sąsiednich iloczynów, aby wyrugować jak największą liczbę literalów. Najważniejsze, by liczba grupowanych sąsiednich iloczynów była potęgą dwójki. Zatem reprezentacja graficzna funkcji przełączającej może posłużyć nam do przeprowadzenia minimalizacji. Przestrzeń trójwymiarowa, służąca dotychczas do ilustracji poprawności naszego rozumowania, nie jest przestrzenią złożoną i pomijając problemy rysunkowe możemy ją wykorzystywać w procesie minimalizacji.



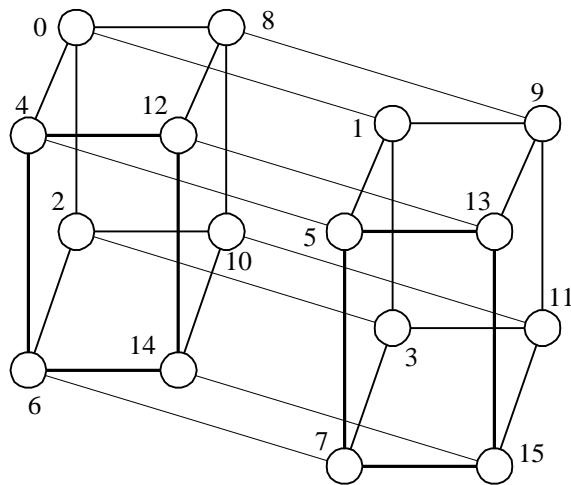
Rys. 5.4 Przykłady graficznych reprezentacji funkcji przełączających

Na rysunku 5.5 przedstawiliśmy reprezentację przestrzeni czterowymiarowej. Zauważmy, że oprócz problemu związanego z jej narysowaniem, znacznie skomplikowało się zagadnienie rysowania w przestrzeni „obwódów” łączących 2^A sąsiednich wierzchołków. Problem potęguje się wraz ze zwiększeniem liczby wymiarów. Właściwie począwszy od przestrzeni czterowymiarowej proces minimalizacji na przestrzeni jest bardzo utrudniony. Dlatego możemy zadać sobie pytanie: czy wykorzystując naturalne własności przestrzeni, szczególnie przydatne do minimalizacji funkcji przełączających, możemy uniknąć kłopotliwego rysowania tych przestrzeni? Najlepiej byłoby znaleźć pewną reprezentację przestrzeni odwzorowaną na płaszczyznę. Oczywiście odwzorowanie to powinno zachować istotne dla nas sąsiedztwo wierzchołków lub zbytnio go nie zniekształcać.

5.3 Tablica Karnaugh

Rozważmy przestrzeń trójwymiarową podaną na rysunku 5.2d. Przetnijmy odcinki łączące wierzchołki oznaczone odpowiednio przez $(\bar{x}_1\bar{x}_2\bar{x}_3, x_1\bar{x}_2\bar{x}_3)$ oraz $(\bar{x}_1\bar{x}_2x_3, x_1\bar{x}_2x_3)$. Tak oswobodzoną przestrzeń rozciągamy na płaszczyźnie. Następnie pozbawiając się odcinków łączących wierzchołki, obrysujemy wierzchołki kratkami przylegającymi bokami do siebie. Otrzymamy tablicę (siatkę), która ma następujące własności:

1. Każda kratka sąsiednia w sensie topologicznym jest sąsiednia w sensie wniosku 5.1.
2. Sąsiednie są także kratki zewnętrzne (ponieważ są to kratki odpowiadające wierzchołkom sąsiednim przestrzeni, które w procesie rozciągania na płaszczyźnie zostały od siebie odsunięte).
3. Każda kratka reprezentuje iloczyn zupełny.



Rys. 5.5 Czterowymiarowa przestrzeń

Opisane powyżej działania zilustrowaliśmy na rys. 5.6. Tak otrzymana tablica może służyć do reprezentacji funkcji przełączającej, ponieważ wpisanie do odpowiednich kratek wartości 1 (a do pozostałych 0, dlatego często kratki te pozostawiamy puste), oznaczać będzie, że wartość funkcji odpowiadająca danemu iloczynowi wynosi 1. Ponadto, tablica ma istotne dla nas sąsiedztwo krutek wykorzystywane przy minimalizacji funkcji. Możemy także stwierdzić, że łatwiej jest operować tablicą niż przestrzenią n-wymiarową. Podobnie możemy postąpić z przestrzenią cztero-, pięcio- czy więcej wymiarową, rysując odpowiadającą im tablicę na płaszczyźnie.

Uważny Czytelnik zauważył już, że w zależności od miejsc, w których dokonujemy przecięcia przestrzeni, możemy otrzymać tablice, różniące się między sobą oznaczeniami pól (krutek). Dlatego, aby uniknąć niepotrzebnego bałaganu, wprowadzimy na potrzeby naszego podręcznika tablice standardowe, które przedstawiliśmy na rys. 5.7.

Dotychczasowe ustalenia mają porządkujący charakter i są właściwie bez znaczenia dla Czytelnika, który dobrze zrozumiał algorytm otrzymywania tablic. Omawiane tablice nazwiemy tablicami Karnaugh (czytaj Karno)[2].

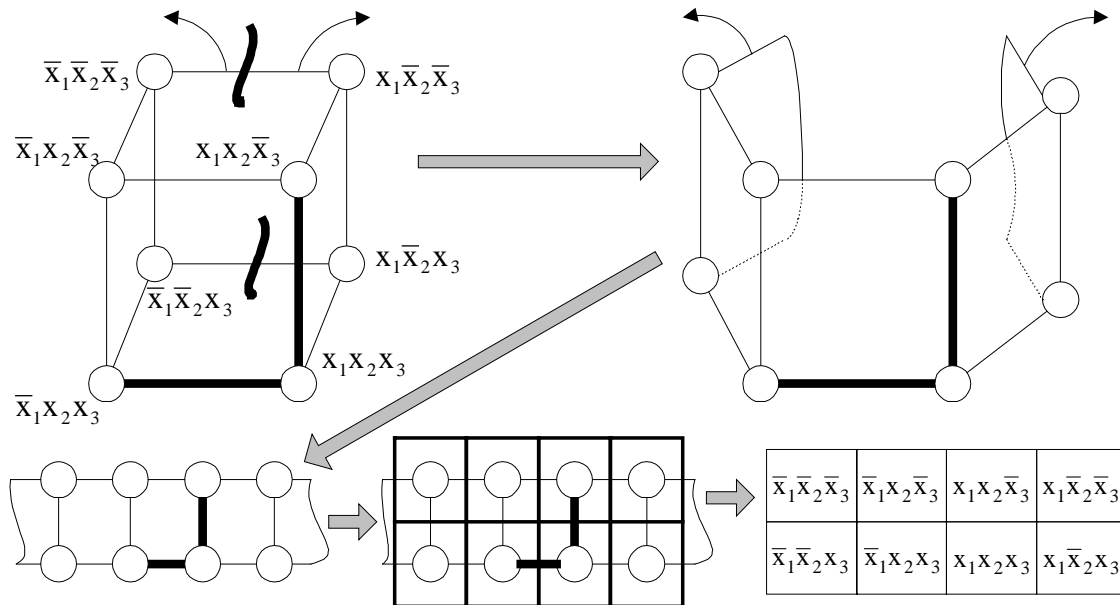
W tablicy czterech zmiennych wszystkie kratki topologicznie sąsiednie oraz kratki zewnętrzne są sąsiednie w sensie wniosku 5.1. Jak łatwo zauważyć, tablica dla funkcji pięciu zmiennych powstała przez połączenie dwóch tablic czterech zmiennych. Warunki sąsiedztwa krutek dla obu tablic czterech zmiennych są identyczne jak dla tablicy pojedynczej (rys. 5.7a). Sąsiednie są również wszystkie te kratki, które tworzą odbicia zwierciadlane w lustrze umiejscowionym na ich styku (rys. 5.7d). Tym samym sąsiednie będą kratki (5,21), (15,31), (2,18) itd.

Dociekliwym Czytelnikom możemy podpowiedzieć, że tablica dla sześciu zmiennych powstanie z dwóch tablic pięciu zmiennych (lustro w środku), siedmiu zmiennych z dwóch tablic sześciu zmiennych itd.

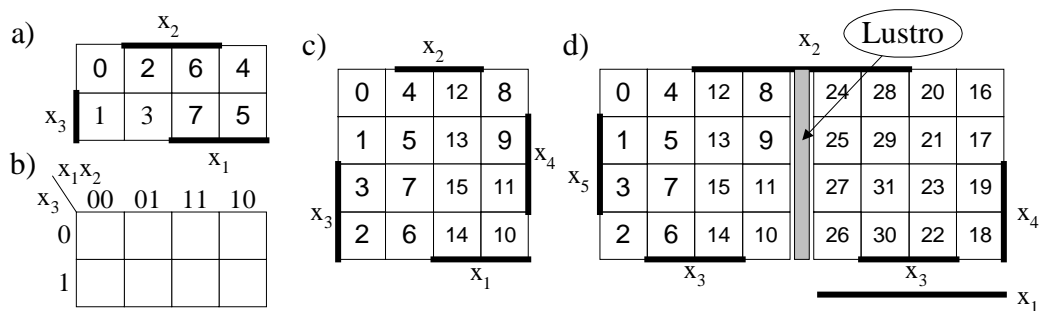
Literatura

[1] Bromirski J., Teoria automatów, WNT, Warszawa 1969,

[2] Karnaugh M., A Map method for Synthesis of Combinational Logic Circuits, Trans. AIEE, Communications and Electronics, 72, str.593-599, Nov. 1953



Rys. 5.6 Ilustracja rozciągnięcia przestrzeni na płaszczyźnie, a - kolejne fazy rozciągania, b,c - wynik rozciągania



Rys. 5.7 Standardowe tablice Karnaugh: a,b - tablice trzech zmiennych, c - tablica czterech zmiennych, d - tablica pięciu zmiennych

ĆWICZENIA

- Podaj przykłady tablic Karnaugh, gdy „przecięcia” przestrzeni trójwymiarowej z rys. 5.6a dokonamy na odcinkach łączących wierzchołki ($\bar{x}_1\bar{x}_2\bar{x}_3$, $\bar{x}_1x_2\bar{x}_3$) i ($\bar{x}_1\bar{x}_2x_3$, $\bar{x}_1x_2x_3$).
- Wpisz do tablicy Karnaugh następujące funkcje przełączające:
 - $f(abc) = \Sigma(0,1,6,7)$,
 - $f(abcd) = \Sigma(0,1,6,7,12,15)$,
 - $f(abcd) = \Sigma(4,5,7,8,9,11,12,15)$,
 - $f(abcde) = \Sigma(0,1,6,7,12,15,30,31)$,
 - $f(abcde) = \Sigma(12,13,17,30,31,45,48,60,61)$.
- Podaj liczbę możliwych tablic Karnaugh dla $n=3, 4$.
- Wpisz do tablicy Karnaugh następujące funkcje przełączające:
 - $f(x_1x_2x_3x_4) = \bar{x}_2\bar{x}_4 + \bar{x}_3\bar{x}_4 + x_2x_3x_4$

$$\text{b) } f(abcd) = \overline{a}bc + \overline{a}c\overline{d} + \overline{a}b\overline{d} + bcd + abc + \overline{a}cd + abd + b\overline{c}d$$

$$\text{c) } f(x_1x_2x_3x_4) = \overline{x}_1x_2\overline{x}_3 + x_1\overline{x}_3\overline{x}_4 + x_1\overline{x}_2x_4 + x_2x_3x_4$$

$$\text{d) } f(x_1x_2x_3x_4) = x_2\overline{x}_3\overline{x}_4 + x_1\overline{x}_2\overline{x}_3 + \overline{x}_1x_2x_4 + x_1x_3x_4$$

$$\text{e) } f(abcd) = \overline{b}d + \overline{c}d + bd + \overline{a}cd$$

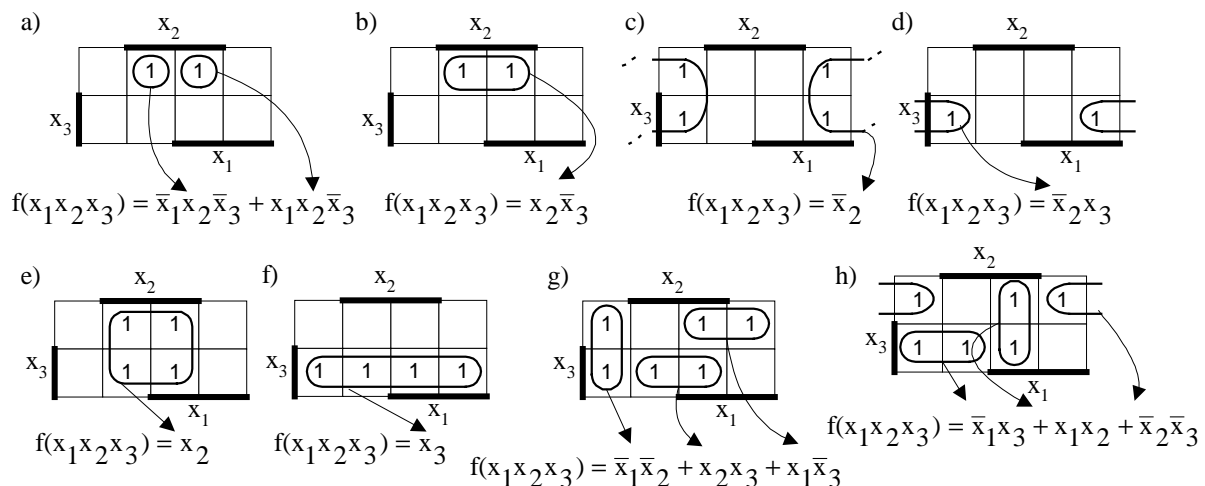
$$\text{f) } f(abcd) = b\overline{c} + \overline{b}d + \overline{a}cd + bd$$

$$\text{g) } f(abcd) = b\overline{c} + \overline{a}bc + \overline{b}d + bd$$

6 MINIMALIZACJA FUNKCJI PRZELĄCZAJĄCYCH

6.1 Minimalizacja funkcji w pełni określonych

W niniejszym rozdziale będziemy wykorzystywać wiadomości z rozdziału 5. Przypomnijmy zatem, że dowolną funkcję przełączającą możemy przedstawić w kanonicznej postaci dysjunkcyjnej. Wiemy także, że istnieje 2^n iloczynów zupełnych n zmiennych. Tablica Karnaugh dla funkcji n zmiennych ma 2^n krutek, z których każda reprezentuje ściśle określony iloczyn zupełny. Kratka może być opisana wprost za pomocą iloczynu zupełnego bądź jego odpowiednika dziesiętnego, albo za pomocą zmiennych opisujących na zewnątrz tablicy wskazane pola. Pole wyznaczone przez dowolną zmienną składa się z 2^{n-1} krutek. Pola częściowo się pokrywają. Po tym krótkim przypomnieniu wiadomości z poprzednich rozdziałów, możemy przystąpić do wpisania dowolnych funkcji przełączających do tablicy Karnaugh. Uczynimy to, przytaczając wiele przykładów ilustrujących sposób wpisywania funkcji do tablic. Będą to przykłady specjalnie dobrane ze względu na swój aspekt dydaktyczny (rys. 6.1).

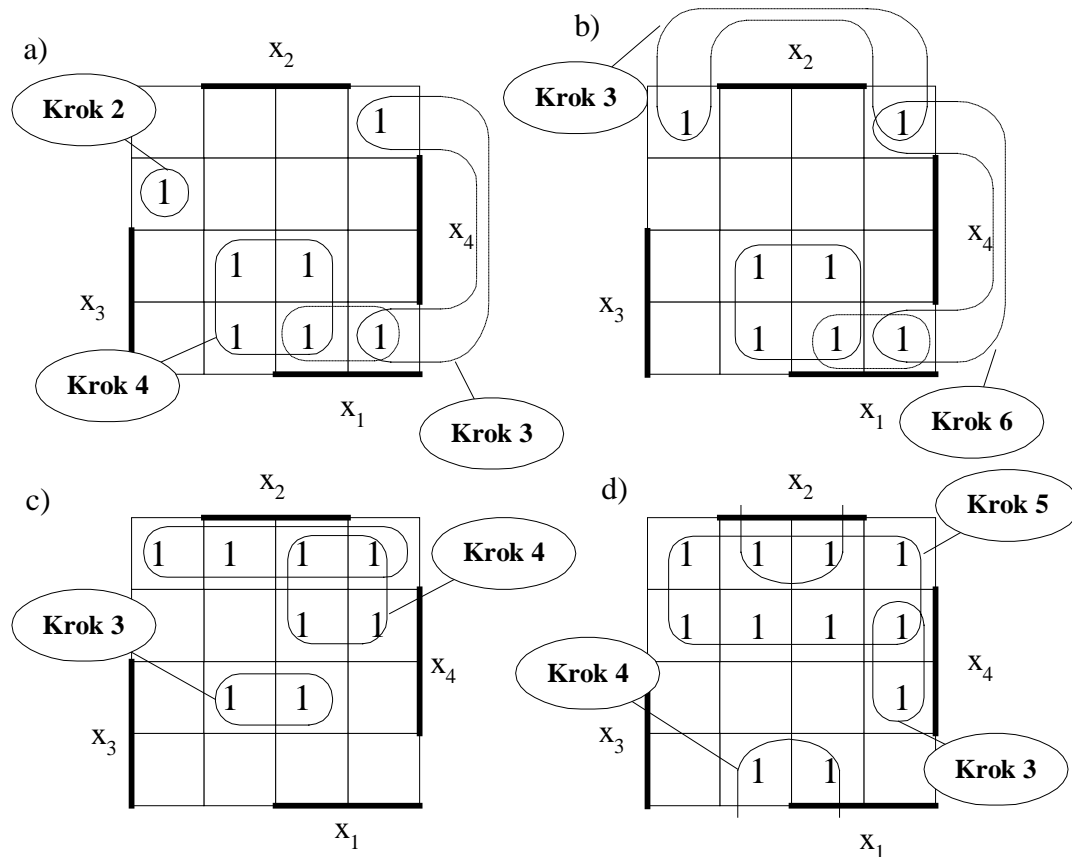


Rys. 6.1 Przykłady ilustrujące zależność między postaciami dysjunkcyjnymi a ich reprezentacją w tablicy Karnaugh

Poniżej przytoczyliśmy szkic metody minimalizacji dowolnej funkcji przełączającej, nazywaną metodą Karnaugh (po raz pierwszy podana w [1] chociaż wcześniej, prawie identyczną metodę, zaprezentowano w [2]).

1. Wpisz funkcję do tablicy.
2. Opisz obwódką pojedyncze jedynki, których nie można połączyć z innymi jedynkami („osamotnione jedynki”),
3. Znajdź jedynkę (jeżeli istnieje), którą można połączyć tylko z jedną sąsiednią jedynką tworząc obwódkę dwukratkową.
4. Znajdź nie obwiedzione jedynki (jeżeli istnieją), które mogą utworzyć obwódkę zawierającą tylko cztery jedynki.
5. Powtórz proces określania obwódek zawierających 8,16,32,... jedynek sąsiednich.
6. Jeżeli po wykonaniu kroku 4 i 5 pozostaną nie obwiedzione jedynki, połącz je z już obwiedzionymi jedynkami w taki sposób, by obwódka ta reprezentowała iloczyn o najmniejszej liczbie literalów.

Na rysunku 6.2 zilustrowaliśmy przykładami powyższy algorytm.



Rys. 6.2 Przykłady ilustrujące metodę minimalizacji za pomocą tablic Karnaugh

Rozważmy funkcję wpisaną do tablicy z rys. 6.2a. Możemy wykazać, wykorzystując w tym celu definicję 3.2.4.3, że implikantami prostymi funkcji są:

$$\bar{x}_1\bar{x}_2\bar{x}_3x_4, x_2x_3, x_1\bar{x}_2\bar{x}_4, x_1x_3\bar{x}_4.$$

Implikant prosty x_2x_3 reprezentowany jest w tablicy Karnaugh przez obwódkę zawierającą cztery jedynki, implikanty proste $x_1\bar{x}_2\bar{x}_4$, $x_1x_3\bar{x}_4$ przez obwódki po dwie jedynki oraz $\bar{x}_1\bar{x}_2\bar{x}_3x_4$ reprezentowany jest przez pojedynczą jedynkę.

Implikanty proste są zawsze reprezentowane przez możliwie największe grupy zawierające 1,2,4,8,... lub 2^k sąsiednich jedynek. Implikantami będą dowolne podzbiory grup największych zawierających 1,2,4,8,... itd. jedynek sąsiednich.

Zgodnie z twierdzeniem 3.2.4.2, każdą funkcję przełączającą możemy przedstawić jako sumę wyłącznie implikantów prostych. Z powyższego przykładu wynika, że nie wszystkie implikanty proste muszą koniecznie wejść w skład postaci dysjunkcyjnej, ponieważ niektóre są nadmiarowe (*redundant*). Takim implikantem jest $x_1x_3\bar{x}_4$ (obwódka przerywana na rys. 6.2a oraz 6.2b). Niezbędnymi implikantami z rys. 6.2a, które muszą wejść w skład postaci dysjunkcyjnej są:

$$\bar{x}_1\bar{x}_2\bar{x}_3x_4, x_2x_3, x_1\bar{x}_2\bar{x}_4.$$

Implikanty te nazywamy zasadniczymi implikantami prostymi (*essential prime implicants*), ponieważ obwódki tych implikantów jako jedyne pokrywają niektóre jedynki funkcji.

Minimalna postać dysjunkcyjna funkcji przełączającej zawiera najmniejszą liczbę składników i literałów, co jest jednoznaczne z umieszczeniem w postaci dysjunkcyjnej zasadniczych implikantów prostych i takich, które pokryją pozostałe jedynki występujące w tablicy Karnaugh'a. Jest to prawda pod warunkiem, że wartość funkcji kosztu jest dodatnia, koszty realizacji układu to suma kosztów wybranych implikantów oraz zawsze, gdy implikant P1 pokrywa P2 to koszt realizacji P1 jest nie większy od kosztu realizacji P2.

Minimalną postacią dysjunkcyjną rozważanej funkcji jest

$$f(x_1x_2x_3x_4) = \bar{x}_1\bar{x}_2\bar{x}_3x_4 + x_2x_3 + x_1\bar{x}_2\bar{x}_4.$$

Podobnie możemy poszukiwać minimalnych postaci koniunkcyjnych. Dla rozważanej przez nas funkcji jej dopełnienie

$$\bar{f}(x_1x_2x_3x_4) = x_2\bar{x}_3 + x_1\bar{x}_2x_4 + \bar{x}_1\bar{x}_3\bar{x}_4 + \bar{x}_1\bar{x}_2x_3$$

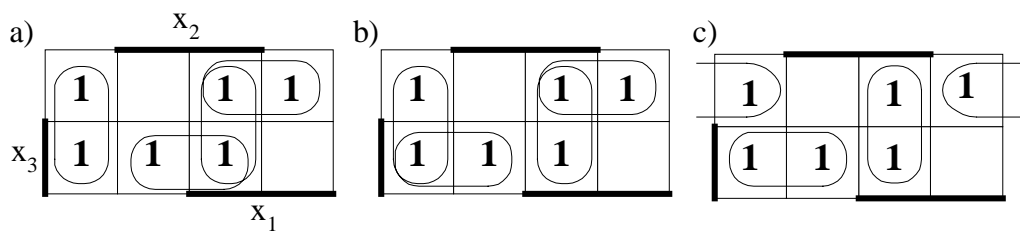
i dalej

$$\bar{\bar{f}} = f = (\bar{x}_2 + x_3) \cdot (\bar{x}_1 + x_2 + \bar{x}_4) \cdot (x_1 + x_3 + x_4) \cdot (x_1 + x_2 + \bar{x}_3).$$

Można stwierdzić, że minimalną postacią danej funkcji jest minimalna postać dysjunkcyjna, zawierająca mniejszą liczbę składników w porównaniu z liczbą czynników postaci koniunkcyjnej oraz mniejszą liczbę literałów (odpowiednio 9 i 11).

Przykład:

Często istnieje kilka możliwości wyboru obwódów, które pokrywają wszystkie jedynki funkcji. Załóżmy, że interesują nas tylko obwódki reprezentujące implikanty proste. Wtedy, dla funkcji z rys. 6.1h – dla wygody powtórzmy ją na rys. 6.3 – możemy zrealizować przykła-



Rys. 6.3 Rodzaje pokryć funkcji: a) proste nadmiarowe, b) proste nienadmiarowe, c) minimalne

dowe pokrycie proste nadmiarowe złożone z wybranych implikantów prostych (rys. 6.3a, bez utraty pokrycia możemy usunąć z niego implikant x_1x_2), pokrycie proste nienadmiarowe (rys.6.3b, nie możemy usunąć żadnego implikantu bez utraty pokrycia) oraz minimalne pokrycie, które zawiera minimalną liczbę implikantów (dla tej funkcji istnieje drugie minimalne pokrycie – porównaj rys. 6.1g). W praktyce zadawalającym rozwiązaniem jest pokrycie proste i nienadmiarowe (np. z punktu widzenia diagnostyki technicznej, ponieważ minimalne pokrycie nie zapewnia testowalności lepszej niż w przypadku pokrycia prostego i nienadmiarowego). Zauważmy, że pokrycie minimalne jest proste i nienadmiarowe, ale nie na odwrót.

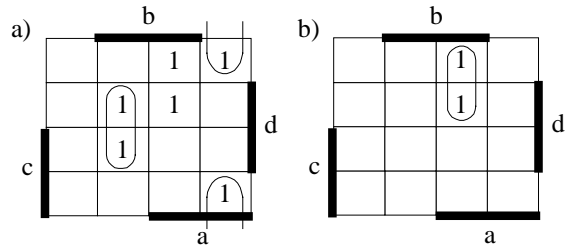
W ogólnym przypadku, gdy rozpoczniemy minimalizację od poszukiwania zasadniczych implikantów i tymczasowo wykreślimy je z tablicy Karnaugh'a to tym prostym zabiegiem zawężymy sobie pole wyboru pozostałych implikantów prostych. Komplikuje się postępowanie w przypadku, gdy funkcja nie posiada zasadniczych implikantów, co postaramy się wyjaśnić na przykładzie odpowiednio dobranej funkcji.

Przykład

Niech będzie dana funkcja $f(abcd) = \sum(2,3,5,7,8,10,12,13)$. Implikantami prostymi funkcji są:

$\bar{a}\bar{b}c$, $\bar{a}cd$, $\bar{a}bd$, $\bar{b}cd$, $\bar{a}bd$, $\bar{a}cd$, $\bar{b}cd$, $\bar{a}b\bar{c}$. Żaden z nich nie jest zasadniczy.

Wybierając arbitralnie implikant prosty $\bar{a}\bar{b}c$ do tworzonej minimalnej postaci dysjunkcyjnej i tymczasowo usuwając z funkcji jedynki przez niego pokrywane otrzymamy dla pozostałych jedynek następujący zbiór implikantów: $\bar{a}bd$, $ab\bar{d}$, $a\bar{c}\bar{d}$, $b\bar{c}d$, abc . Dwa pierwsze są implikantami zasadniczymi (nazywane na tym etapie pseudozasadniczymi, rys. 6.4a), ponieważ pierwszy jako jedyny pokrywa jedynkę funkcji dla iloczynu zupełnego $\bar{a}\bar{b}c\bar{d}$, a drugi dla iloczynu $\bar{a}bcd$. Jeżeli podobnie usuniemy z funkcji wszystkie jedynki, które te dwa implikanty pokrywają to pozostanie nam jedyny wybór jakim jest implikant abc (rys. 6.4b). Zatem minimalne pokrycie jest następujące: $\bar{a}\bar{b}c$, $\bar{a}bd$, $ab\bar{d}$, abc .



Rys. 6.4 Implikanty pseudozasadnicze

Zachęca się rozpocząć minimalizację od arbitralnego wyboru implikantu $\bar{a}cd$. Czy otrzymaliście trzy następujące implikanty: $\bar{b}c\bar{d}$, $b\bar{c}d$ (pseudozasadnicze) oraz $a\bar{c}\bar{d}$ (dopełniający pokrycie), co daje kolejne minimalne pokrycie funkcji: $\bar{a}cd$, $\bar{b}c\bar{d}$, $b\bar{c}d$, $a\bar{c}\bar{d}$?

Można uzyskiwać jeszcze lepsze rezultaty stosując faktoryzację. Polega ona na wyłączeniu przed nawias wspólnych części iloczynów. Dla przykładu funkcja $f(x_1x_2x_3) = x_2x_3 + x_1x_2$ jest minimalną postacią dysjunkcyjną. Wyłączając przed nawias x_2 , otrzymamy $f(x_1x_2x_3) = x_2(x_3 + x_1)$, eliminując w realizacji układowej jedną bramkę AND.

Niekiedy prościej jest zminimalizować funkcję \bar{f} (dopełnienie) oraz zrealizować ją układowo z dołączoną bramką NOT na wyjściu (w urządzeniach programowalnych istnieje standardowa możliwość dopełniania wyjść).

Metoda minimalizacji za pomocą tablicy Karnaugh'a jest prosta i powszechnie stosowana przy niewielkiej liczbie zmiennych (praktycznie do pięciu zmiennych), zaś sama tablica jest także wygodnym narzędziem dydaktycznym umożliwiającym przedstawienie własności funkcji przełączających, dlatego będziemy ją chętnie stosować.

Do minimalizacji funkcji o liczbie zmiennych większej niż pięć stosuje się inne metody, o których powiemy w następnych rozdziałach.

Na koniec niniejszego rozdziału warto zapytać: Czy w dobie wielkiego rozwoju technologicznego układów scalonych opłaca się minimalizować funkcje przełączające, oszczędzając na realizacji układowej, jeżeli koszt samego procesu minimalizacji jest olbrzymi? Odpowiedź jest twierdząca, ponieważ z punktu widzenia projektanta układu scalonego o potrzebie minimalizacji funkcji decydują inne czynniki niż tylko „oszczędność realizacyjna”. Poniżej wspomnimy tylko o wybranych.

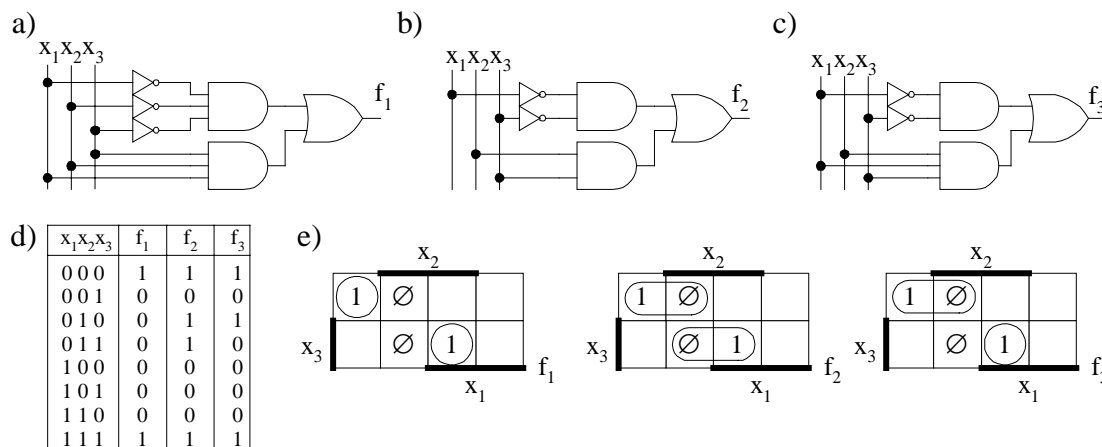
Obecnie kilkadziesiąt procent powierzchni układu scalonego zajmuje metalizacja, nieodłączna część połączeń międzyelementowych. Jeżeli zmniejszymy liczbę tych połączeń, to znacznie zmniejszymy obszar zajmowany przez metalizację, a w miejscu zwolnionym można będzie umieścić elementy aktywne (tranzystory), ponieważ nietrudno zauważyć, że proces minimalizacji redukuje także liczbę połączeń międzyelementowych.

W układach minimalnych (a właściwie prostych i nienadmiarowych) łatwiej wykrywać i lokalizować uszkodzenia logiczne. Jest pożądane, by można było wykrywać jak największą liczbę uszkodzeń, czyli zapewnić jego całkowitą testowalność (patrz rozdział o diagnostyce technicznej).

Stosowanie programowalnych urządzeń logicznych (PAL, PLA), także wymaga minimalizacji funkcji przełączających, aby „zmieścić” ją w urządzeniu (ograniczona jest dostępna liczba bramek AND i OR).

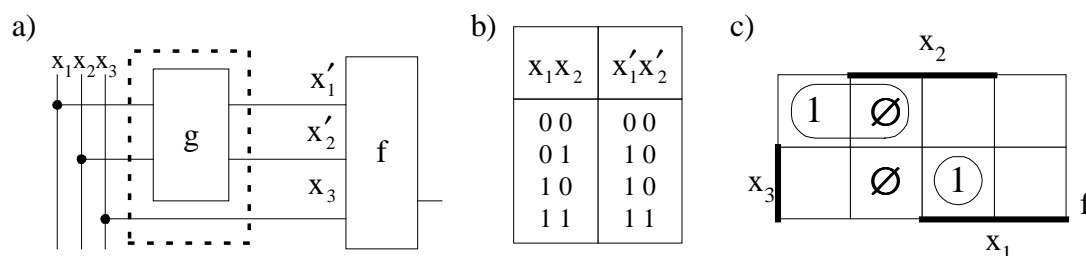
6.2 Minimalizacja funkcji nie w pełni określonych

Przypomnijmy, że funkcją nie w pełni określoną nazwaliśmy funkcję, w odniesieniu do której nie określono wartości dla wybranych iloczynów zupełnych. Innymi słowy, dla zadanej funkcji w jej kanonicznej postaci dysjunkcyjnej podajemy wszystkie iloczyny zupełne, nie-które pomijamy, a wybór pozostałych pozostawiamy jako opcjonalny (mogą wejść do postaci kanonicznej, ale nie muszą). Opcjonalne iloczyny zupełne nazywamy obojętnymi (*don't care*). Na tablicy Karnauga iloczyny obojętne oznaczamy przez \emptyset .



Rys. 6.5 Minimalizacja funkcji z wykorzystaniem iloczynów obojętnych:
a,b,c - realizacje funkcji f_1, f_2, f_3 , d - tablica prawdy, e - tablice Karnauga

Weźmy pod uwagę trzy układy przedstawione na rys. 6.5. Zauważmy, że jeżeli na wejścia układów nigdy nie podamy zestawu wartości zmiennych wejściowych $x_1 x_2 x_3 = 010$ oraz $x_1 x_2 x_3 = 011$, to wartości funkcji f_1, f_2, f_3 będą identyczne (patrz tablica prawdy), pomimo że realizowane są one przez trzy układy różne pod względem złożoności. Minimalny jest układ z rys. 6.5b.



Rys. 6.6 Przykład ilustrujący wykorzystanie iloczynów obojętnych: a - struktura układu,
b - tablica prawdy dla dwuwyjściowego układu g , c - iloczyny obojętne funkcji f

Na rysunku 6.5e przedstawiliśmy tablice Karnauga dla funkcji f_1, f_2, f_3 zaznaczając, w jaki sposób iloczyny obojętne wykorzystujemy na etapie minimalizacji funkcji. Reguły te są już nam znane. Musimy jednak pamiętać, że w obwodce nie mogą znajdować się tylko iloczyny obojętne. Jeżeli wykorzystujemy iloczyny obojętne (niekoniecznie wszystkie), to w obwodce musi się znaleźć przynajmniej jedna jedynka (Dlaczego?).

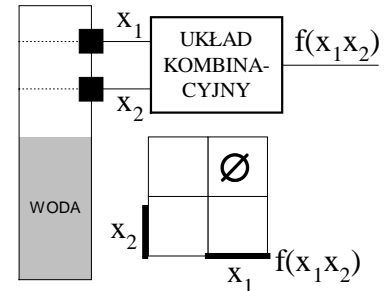
Przykład:

Rozważmy układ kombinacyjny przedstawiony na rys. 6.6a. Układ kombinacyjny oznaczony literką g jest układem dwuwejściowym realizującym funkcje zgodnie z tablicą prawdy z rys. 6.6b.

Nietrudno zauważyć, że na wyjściach $x'_1x'_2$ nie pojawią się nigdy wartości $x'_1x'_2=01$. Układ ten przekształca $x_1x_2=01 \rightarrow x'_1x'_2=10$, nie zmieniając pozostałych wartości. Innymi słowy, na wejście układu f nigdy nie zostaną podane wartości $x'_1x'_2x_3=010$ oraz 011 , czyli istnieją dwa iloczyny obojętne, które możemy wykorzystać na etapie minimalizacji układu realizującego funkcje $f(x'_1x'_2x_3)$.

Przykład:

Na rysunku 6.7 przedstawiliśmy szyb kopalniany z umieszczonymi w nim czujnikami poziomu wody x_1 oraz x_2 . Gdy poziom wody podniesie się i osiągnie j -ty czujnik, wtedy $x_j=1$, $1 \leq j \leq 2$. Wysokość słupa wody ulega wahaniom. Nietrudno zauważyć, że na wejście układu kombinacyjnego nigdy nie zostaną podane wartości $x_1x_2=10$. Dlatego iloczynem obojętnym jest $x_1\bar{x}_2$.



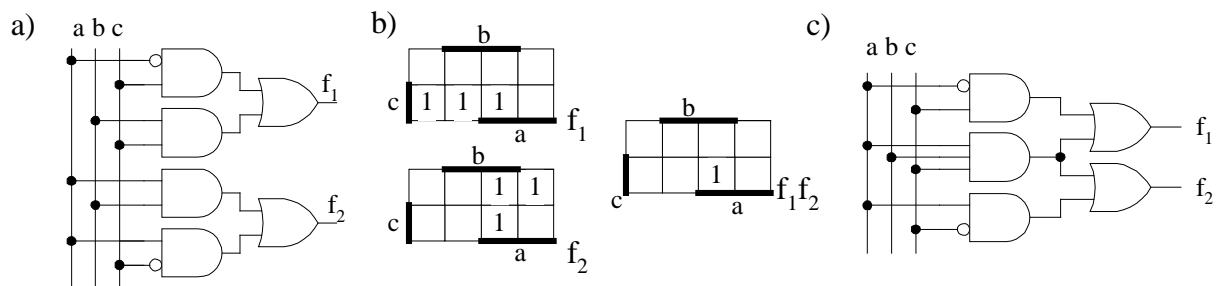
Rys. 6.7 Praktyczna ilustracja wykorzystania iloczynów obojętnych

6.3 Minimalizacja zespołu funkcji przełączających

W poprzednich rozdziałach zajmowaliśmy się minimalizacją pojedynczej funkcji przełączającej. Dla danej funkcji podawaliśmy minimalny jednowyjściowy układ logiczny. W wielu zagadnieniach występują dwie lub więcej funkcji przełączających, dla których należy podać realizację układową. Układ kombinacyjny realizujący kilka funkcji przełączających nazywa się układem wielowyjściowym. Załóżmy, że układ ma realizować dwie funkcje: f_1 oraz f_2 . Jeżeli zestawimy dwa układy jednowyjściowe - jeden realizujący funkcję f_1 , a drugi f_2 - otrzymamy układ dwuwyjściowy. Czy ten sposób rozwiązania zagadnienia jest dobry? Niech odpowiedzią na pytanie będzie kolejny przykład.

Przykład:

Niech będą dane dwie funkcje $f_1(abc) = \sum(1,3,7)$ oraz $f_2(abc) = \sum(4,6,7)$. Minimalne układy



Rys. 6.8 Układy dwuwyjściowe: a - realizacja funkcji f_1 i f_2 przez pojedyncze układy (6 bramek), b - tablice Karnaugh dla funkcji f_1 , f_2 i f_1f_2 , c - realizacja dwuwyjściowa (5 bramek)

dy realizujące z osobna każdą z funkcji zawierają w sumie 6 bramek (rys. 6.8a). Rozważmy tablice Karnaugh dla funkcji f_1 , f_2 oraz $f_1 \cdot f_2$ (rys. 6.8b). Szczególnie ważna jest tablica z wpisaną funkcją $f_1 \cdot f_2$ (iloczyn funkcji f_1 oraz f_2). Zauważmy, że iloczyn abc jest wspólny dla funkcji f_1 oraz f_2 , może zatem być wykorzystany do realizacji obu funkcji, chociaż nie jest implikantem prostym ani funkcji f_1 ani f_2 . Zatem minimalną realizacją dwuwyjściową funkcji f_1 oraz f_2 będzie układ z rys. 6.8c, zawierający 5 bramek.

Jeżeli pewien układ logiczny ma trzy wyjścia, to musimy rozważyć trzy funkcje f_1, f_2, f_3 . Postępując analogicznie jak w przykładzie powyżej, należy wziąć pod uwagę następujące iloczyny funkcji: $f_1 \cdot f_2, f_2 \cdot f_3, f_1 \cdot f_3$ oraz $f_1 \cdot f_2 \cdot f_3$. Rozważmy zatem kolejny przykład.

Przykład:

Niech będą dane trzy funkcje:

$$f_1(abcd) = \sum(12,13,14,15), \quad f_2(abcd) = \sum(12,14), \quad \text{oraz} \quad f_3(abcd) = \sum(13,15)$$

dla których,

$$f_1 \cdot f_2 = \sum(12,14) = f_2$$

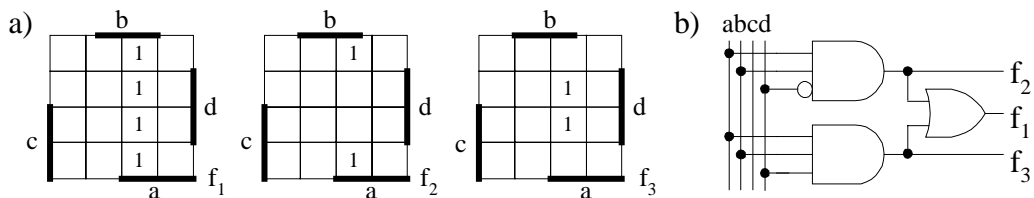
$$f_1 \cdot f_3 = \sum(13,15) = f_3$$

$$f_2 \cdot f_3 = 0$$

$$f_1 \cdot f_2 \cdot f_3 = 0$$

Zauważmy ponadto, że $f_1 = f_2 + f_3$. Układ realizujący powyższe trzy funkcje przedstawiliśmy na rys. 6.9b.

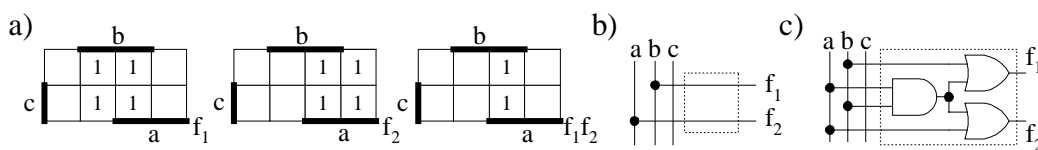
Kolejny przykład będzie pokazywał, że nie należy bezkrytycznie stosować powyższej metody.



Rys. 6.9 Przykład układu trójwyjściowego: a - tablice Karnaugh, b - minimalny układ

Przykład:

Niech będą dane dwie funkcje: $f_2(abc) = \sum(4,5,6,7)$, $f_1(abc) = \sum(2,3,6,7)$. Realizując każdą z funkcji z osobna otrzymamy układ z rys. 6.10b, a stosując powyższą metodę otrzymamy realizację układową z rys. 6.10c.



Rys. 6.10 Przykład realizacji układu dwuwyjściowego: a - tablice Karnaugh dla funkcji f_1, f_2, f_1f_2 , b - realizacja funkcji f_1 i f_2 przez pojedyncze układy (brak bramek), c - układ realizujący f_1 i f_2 (3 bramki)

Wykorzystanie wspólnych iloczynów do uproszczenia realizacji układu wielowyjściowego daje zatem dobre rezultaty, jeżeli iloczyn ten jest implikantem prostym dwóch lub większej liczby funkcji wyjściowych (najlepiej, gdy jest on zasadniczym implikantem prostym).

Podsumowując, rozważmy jeszcze jeden przykład funkcji nie w pełni określonych.

Przykład:

Niech będą dane następujące dwie funkcje:

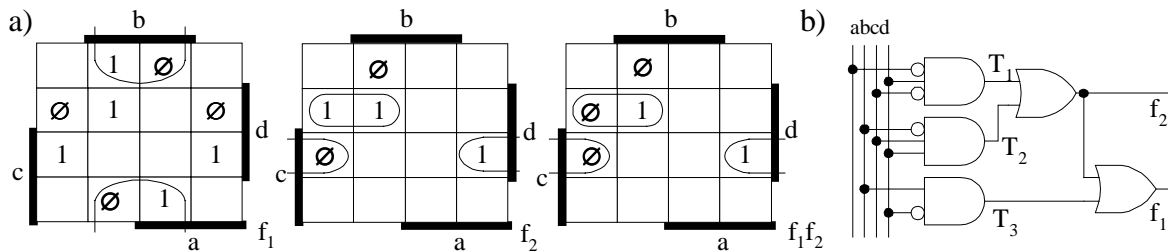
$$f_1(abcd) = \sum(3,4,5,11,14) + \sum \emptyset(1,6,9,12)$$

$$f_2(abcd) = \sum(1,5,11) + \sum \emptyset(3,4),$$

które przedstawiliśmy na rys. 6.11a.

Jeżeli dla funkcji $f_1 \cdot f_2$ wybierzemy implikanty proste $T_1 = \bar{a} \cdot \bar{c} \cdot d$ i $T_2 = \bar{b} \cdot c \cdot d$, to pokryją one wszystkie jedynki funkcji f_2 oraz jedynki funkcji f_1 leżące w kratkach $\{1,3,5,11\}$. Implikanty te wykorzystamy jako wspólne do realizacji funkcji f_1 oraz f_2 .

Minimalną realizację układu dwuwyjściowego przedstawiliśmy na rys. 6.11b.



Rys. 6.11 Przykład realizacji układu dwuwyjściowego : a - tablice Karnaugh'a dla funkcji f_1 , f_2 oraz f_1f_2 , b - minimalna realizacja układu

6.4 Algorytmiczne metody minimalizacji funkcji

Metoda minimalizacji wykorzystująca tablice Karnaugh'a jest praktycznie przydatna dla funkcji do pięciu zmiennych. Liczba zmiennych w rzeczywistych układach jest znacznie większa, zatem należy poszukiwać innych efektywnych metod.

6.4.1 Metoda Quine'a-McCluskey'a

Metodę Karnaugh'a można oprogramować np. dla celów dydaktycznych. Problemy się pojawiają, gdy chcielibyśmy narysować na ekranie monitora tablicę dla $n=16$, nie wspominając już o mozolnym jej wypełnianiu wartościami funkcji.

Pierwszą algorytmiczną metodę minimalizacji funkcji podał Quine [4], którą ulepszył McCluskey [3] dlatego nazywana jest teraz metodą Quine'a-McCluskey'a (w skrócie QM).

W metodzie QM wyróżnia się dwa zasadnicze kroki:

1. generujemy wszystkie proste implikanty funkcji,
2. wybieramy tylko te implikanty proste, które wchodzi do postaci minimalnej (tworzą minimalne pokrycie).

Do generacji implikantów prostych stosowana jest zależność:

$$Ax_i + A\bar{x}_i = A \quad (6.1)$$

gdzie A jest implikantem funkcji, który nie zawiera x_i ani \bar{x}_i .

Jeżeli zależność 6.1 zastosujemy do wszystkich par z listy iloczynów zupełnych danej funkcji to otrzymamy implikanty proste. Celem polepszenia skuteczności procesu generowania implikantów prostych stosuje się pewną notację dla implikantów (kostki). W notacji tej każdą zmienną afirmowaną zastępujemy przez 1, dopełnienie przez 0 a zmienną nie występującą w iloczynie zastępujemy przez 2. Przykładowo trzy implikanty: abc , $\bar{a}\bar{b}\bar{c}$ i ac zastępujemy odpowiednio przez kostki 111, 100 i 121. Jeżeli dwa implikanty spełniają zależność 6.1, to kostki różnią się dokładnie na jednej pozycji i pozycja ta nie jest równa 2. Zatem, gdy iloczyny zupełne funkcji pogrupujemy ze względu na liczbę jedynek, czyli grupa I – kostki z

jedną jedynką, grupa II – kostki z dwoma jedynkami, grupa III – kostki z trzema jedynkami, itd., wtedy sprawdzenie zależności 6.1 dotyczyć będzie par implikantów pochodzących z sąsiednich grup (grupy I i II, II i III itd.). Spełnienie zależności 6.1 zaznaczamy gwiazdką (*) obok implikantów grupy k i $k+1$, a powstały implikant A (z 2 na pozycji, gdzie się różniły kojarzone implikanty) zapisujemy w osobnej kolumnie. Oznaczony gwiazdką implikant grupy k bierze udział w kolejnych sprawdzeniach zależności 6.1. Analogicznie postępujemy z nowopowstałymi kostkami, z tym, że w tym przypadku kojarzymy tylko takie kostki, w których 2-ki występują na tych samych pozycjach (np. 2110 skojarzymy z 2111).

Jeżeli wszystkie pary sąsiednich grup są sprawdzone, to identycznego sprawdzenia dokonujemy dla utworzonej listy, aż do momentu, gdy żadna nowa lista implikantów nie może być utworzona.

Przykład:

Jak już wspomnieliśmy algorytm rozpoczyna się od listy iloczynów zupełnych, dlatego rozważmy przykładową funkcję daną w postaci dziesiętnej: $f(abcd)=\Sigma(1,6,7,8,10,14,15)$. Pogrupowaną listę kostek odpowiadających iloczynom zupełnym przedstawiliśmy na rys. 6.12a.

a)	abcd	b)	abcd	c)	abcd
grupa 1	1 0001 8 1000*	grupa 5	8,10 1020	grupa 8	6,7,14,15 2112
grupa 2	6 0110* 10 1010*	grupa 6	6,7 0112* 6,14 2110* 10,14 1210		
grupa 3	7 0111* 14 1110*	grupa 7	7,15 2111* 14,15 1112*		
grupa 4	15 1111*				

Rys. 6.12 Algorytm Quine'a-McCluskey'a

W pierwszym kroku sprawdzamy spełnienie zależności (6.1) porównując pary z grup 1 i 2, 2 i 3 oraz 3 i 4 otrzymując nową listę implikantów (grupy 5, 6 oraz 7). Powtarzamy proces dla grup 5 i 6 oraz 6 i 7 otrzymując grupę 8.

Implikant 2112 z grupy 8 jest także wynikiem połączenia implikantów 6, 14, 7, 15, który na rys. 6.12c został pominięty. Implikanty ze wszystkich list, nie oznakowane * są implikantami prostymi, czyli $\{0001, 1020, 1210, 2112\} = \{\bar{a}\bar{b}\bar{c}d, \bar{a}b\bar{d}, a\bar{c}\bar{d}, bc\}$.

Końcowym etapem algorytmu jest znalezienie minimalnego pokrycia funkcji przez implikanty proste, czyli wybranie z listy takiej ich minimalnej liczby, która pokryje wszystkie jedynki funkcji. W tym celu tworzymy tablice implikantów prostych, której wiersze reprezentują otrzymane implikanty proste a kolumny iloczyny zupełne funkcji. Dla rozważanego przykładu tablicę implikantów prostych przedstawiliśmy na rys. 6.13. Gwiazdką oznaczyliśmy te implikanty, które jako jedyne pokrywają iloczyny zupełne 1, 6, 7, 8 oraz 15, czyli one muszą wejść w skład minimalnej postaci dysjunkcyjnej, ponieważ są zasadniczymi implikantami funkcji. Implikanty zasadnicze pokrywają iloczyny 1, 6, 7, 8, 10, 14, 15 czyli wszystkie, które powinny być pokryte. Zatem minimalne pokrycie składa się z implikantów I_1 , I_2 oraz I_4 .

		1	6	7	8	10	14	15
Implikanty proste	I_1	X						
	I_2				X	X		
	I_3					X	X	
	I_4		X	X			X	X

Rys. 6.13 Tablica implikantów prostych

Decyzję o wyborze implikantów prostych do minimalnego pokrycia możemy także podjąć na podstawie pewnego wyrażenia boolowskiego. Sposób tworzenia tego wyrażenia, opiera się na następującym rozumowaniu: do pokrycia należy wziąć implikant I_1 oraz I_4 oraz I_2 (rys.

6.13, tylko one pokrywają iloczyny 1, 6, 7, 8, 15, w kolumnach występuje jeden X) oraz implikant I_2 lub I_3 oraz I_3 lub I_4 co odpowiada następującemu wyrażeniu:

$$I_1 I_4 I_2 (I_2 + I_3) (I_3 + I_4) = 1$$

Po wymnożeniu lewej strony równania i eliminacji pokrywanych iloczynów otrzymamy:

$$I_1 I_4 I_2 = 1$$

Co oznacza, że do minimalnej postaci należy wybrać implikant I_1 oraz I_2 oraz I_4 .

Proces szukania minimalnego pokrycia nie jest taki prosty jak w rozważanym wyżej przykładzie. Nie zawsze istnieją implikanty zasadnicze, czyli początkowy wybór implikantów nie jest jednoznaczny, bo nie wiemy, który z równoprawnych implikantów wybrać.

Przykład:

Rozważmy funkcję $f = \Sigma(0,1,3,4,6,7)$. Po zastosowaniu algorytmu QM otrzymamy tablicę implikantów prostych, którą przedstawiliśmy na rys. 6.14a. Funkcja ta nie ma zasadniczych implikantów prostych, ponieważ w żadnej z kolumn nie występuje jeden X (jak zauważamy, w każdej kolumnie są dwa X). W takich przypadkach – dosyć typowych dla większości praktycznych funkcji – musimy arbitralnie wybrać jeden z implikantów, np. I_1 albo wyboru dokonać opierając się na pewnej funkcji kosztu związanej z realizacją implikantu. Wykreślamy z tablicy wiersz I_1 oraz kolumny 0 i 1 (rys. 6.14b). Pozostałą część tablicy traktujemy jako nową tablicę implikantów (rys. 6.14c), dla której stosujemy operację usuwania pokrytych implikantów. Na przykład, implikant I_5 jest pokryty przez I_3 , ponieważ wszystkie X-y implikanta I_5 są zawarte w implikancie I_3 . Podobnie jest z I_2 , który pokryty jest przez I_4 . Po wykreśleniu pokrytych implikantów (rys. 6.14

c) identyfikujemy dwa implikanty pseudozasadnicze I_3 i I_4 , które pokrywają wszystkie pozostałe na tym etapie iloczyny zupełne 3,4,6, 7 (rys. 6.14d).

a)

	0	1	3	4	6	7
I_1 0,1	X	X				
I_2 0,4	X			X		
I_3 3,7			X			X
I_4 4,6				X	X	
I_5 1,3		X	X			
I_6 6,7					X	X

b)

	3	4	6	7
I_2 0,4	X		X	
I_3 3,7			X	X
I_4 4,6		X	X	
I_5 1,3	X	X		
I_6 6,7			X	X

c)

	3	4	6	7
I_3 3,7	X		X	
I_4 4,6		X	X	
I_5 1,3	X	X		
I_6 6,7			X	X

d)

	3	4	6	7
I_3 3,7	X		X	
I_4 4,6		X	X	

Rys. 6.14 Tablica implikantów prostych

Podsumowując, wybraliśmy następujące implikanty proste: I_1 , I_3 oraz I_4 . Jeżeli dla omawianego tutaj przykładu funkcji, utworzymy równanie boolowskie:

$$(I_1 + I_2)(I_1 + I_5)(I_3 + I_5)(I_2 + I_4)(I_4 + I_6)(I_3 + I_6) = 1$$

to po wymnożeniu i uproszczeniach otrzymamy:

$$I_1 I_3 I_4 + I_2 I_5 I_6 + I_2 I_3 I_4 I_5 + I_1 I_2 I_3 I_6 + I_1 I_4 I_5 I_6 = 1$$

Zatem minimalne pokrycia są dwa: I_1 , I_3 i I_4 (otrzymane wyżej) oraz I_2 , I_5 i I_6 .

Funkcje nie w pełni określone minimalizowane są analogicznie:

- iloczyny obojętne są traktowane jako normalne iloczyny zupełne (dla których $f=1$),
- tablica implikantów zawiera tylko te kolumny, które reprezentują normalne iloczyny.

Stosując algorytm QM dla funkcji nie w pełni określonych, może się okazać, że pewne wiersze tablicy implikantów mogą nie zawierać żadnego X. Oznacza to, że „pusty” wiersz reprezentuje implikant złożony z samych iloczynów obojętnych. Takich implikantów nie uwzględniamy w postaci dysjunkcyjnej, bo tylko podnoszą koszt realizacji układowej (Dlaczego?).

Wynikiem metody QM jest lista **wszystkich** implikantów prostych, co w niektórych źródłach uznawane jest za zaletę. Zaleta ta może stać się „wadą” w przypadku funkcji, które posiadają znacznie więcej implikantów prostych niż iloczynów zupełnych, np. funkcja *exam* [7], która ma 625 iloczynów zupełnych, posiada 6139 implikantów prostych oraz żadnego implikantu zasadniczego.

Podstawową niedogodnością metody jest konieczność wygenerowania początkowej listy wszystkich iloczynów zupełnych funkcji. Widać ją szczególnie w przypadku, gdy funkcja jest zadana w postaci sumy implikantów prostych (lecz my o tym nie wiemy), wtedy algorytm wymusza przekształcenie ich na iloczyny zupełne, wygenerowanie listy wszystkich implikantów i powtórne wybranie tych z minimalnym pokryciem funkcji, być może tych samych, za pomocą których zadawaliśmy funkcję. Pamiętanie kolejnych list skojarzonych par implikantów też nie jest dogodne.

6.4.2 Metoda iteracyjnego konsensusu

Quine [5] zaproponował inne podejście, które jak można się spodziewać, różni się od metody QM punktem startowym. Metoda znana pod nazwą iteracyjnego konsensusu (*iterative consensus*), rozpoczyna się od implikantów funkcji (mogą to być iloczyny zupełne, implikanty proste lub inne implikanty). Dalej będziemy ją oznaczać skrótem IT.

Nazwa metody pochodzi od iteracyjnego stosowania zależności:

$$Ax + B\bar{x} = Ax + B\bar{x} + AB \quad (6.2)$$

gdzie A i B są iloczynami nie zawierającymi literału x ani \bar{x} .

Iloczyn AB nazywamy konsensem. Zwróćmy uwagę na to, że konsensus może być implikantem (Dlaczego?), może być równy 0 albo jest pusty (nie istnieje).

Przykład

Wykorzystamy skrót Kons(A,B) do oznaczenia konsensusu dwóch implikantów A,B. Poniżej przedstawiliśmy kilka przykładów konsensusu:

$$\begin{array}{lll} \text{Kons}(0102,1022)=0 & \text{Kons}(1002,2101)=1201 & \text{Kons}(1022,2101)=1201 \\ \text{Kons}(1002,1201)=\text{pusty} & \text{Kons}(1022,1201)=\text{pusty} & \text{Kons}(2122,2022)=2222 \end{array}$$

Czy można podać algorytm pozwalający bezpośrednio z kostek otrzymać konsensus zgodny z równaniem 6.2?

W metodzie IT będzie wykorzystywana także zależność (twierdzenie o pokryciu)

$$A+AB=A \quad (6.3)$$

do usuwania pokrytych implikantów. Z dwóch implikantów A oraz AB pozostawiamy A, ponieważ A pokrywa AB.

Przykład

Poniżej podaliśmy przykłady implikantów spełniających relacje pokrywania:

$$1002 < 1022, \quad 0200 < 0220, \quad 1002 < 1022.$$

Można pokazać, że $Y > X$, wtedy i tylko wtedy, gdy

- a) $(\forall x_i \in \{0,1\}, y_i=x_i, \text{ lub } y_i=2),$
- b) $\forall x_i=2, y_i=2.$

Metoda IT to iteracyjne wykonanie następujących dwóch kroków:

Krok 1. Usuń z postaci dysjunkcyjnej wszystkie pokryte implikanty,

Krok 2. Wygeneruj wszystkie (niepuste i różne od 0) konsensusy z par iloczynów. Dodaj je do postaci dysjunkcyjnej. Przejdź do kroku 1.

Algorytm kończy się w momencie, gdy nie możemy wygenerować nowych konsensusów, ponieważ uzyskane iloczyny to implikanty proste.

Przykład:

Niech będzie dana funkcja $f(abcd)=\Sigma(1,6,7,8,10,14,15)$ (identyczna do tej, którą wykorzystaliśmy przy prezentacji algorytmu QM). Zastosujmy tym razem metodę IT do wygenerowania implikantów prostych tej funkcji. Przedstawmy ją w postaci dysjunkcyjnej: $f(abcd) = \bar{a}\bar{b}\bar{c}d + \bar{a}bc + abc + ab\bar{d}$.

Krok 1. Nie ma implikantów pokrywanych przez inne implikanty.

Krok 2. $\bar{a}bc$ oraz abc tworzą konsensus bc , który dodajemy do postaci dysjunkcyjnej, otrzymując: $f(abcd) = \bar{a}\bar{b}\bar{c}d + \bar{a}bc + abc + ab\bar{d} + bc$.

Krok 1'. Konsensus bc pokrywa $\bar{a}bc$ oraz abc , dlatego usuwamy je z postaci dysjunkcyjnej otrzymując: $f(abcd) = \bar{a}\bar{b}\bar{c}d + ab\bar{d} + bc$

Krok 2'. $ab\bar{d}$ oraz bc tworzą konsensus $a\bar{c}d$, który dodajemy do postaci dysjunkcyjnej, otrzymując $f(abcd) = \bar{a}\bar{b}\bar{c}d + ab\bar{d} + bc + a\bar{c}d$.

Krok 1'' Nie ma implikantów pokrywanych przez inne implikanty.

Krok 2'' Nie można wygenerować konsensusu różnego od pustego bądź różnego od 0. STOP.

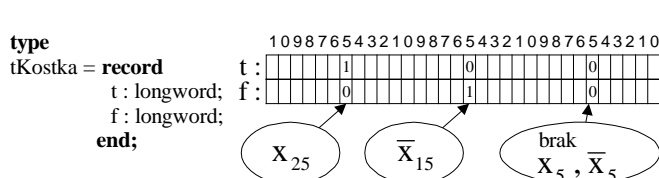
W powyższej postaci dysjunkcyjnej występują wszystkie implikanty proste danej funkcji.

Sprawdź ile należy wykonać powyższych kroków, gdy funkcja będzie zadana w kanonicznej postaci dysjunkcyjnej.

Metoda QM, ze względu na konieczność rozpoczęcia algorytmu od listy iloczynów zupełnych, dużą liczbę iteracji tworzących kolejne listy implikantów i konieczność ich pamiętania jest stosunkowo mało efektywna. Bardziej efektywną jest metoda IT (Dlaczego?, może odpowiedzią jest poniższa próba implementacji metody).

Przystępując do oprogramowania metody IT, należy zdecydować się na taką reprezentację implikantów (kostek), która pozwoli na prosty i efektywny sposób realizacji operacji pokrywania i obliczania konsensusów.

Interesującą propozycję zakodowania kostek podano w pracy[6]. Implikanty reprezentowane są przez rekord składający się z dwóch słów 32-bitowych.



Rys. 6.15 Reprezentacja implikantu (kostki).

Afirmowaną zmienną występującą w implikancie reprezentujemy przez wartość $t=1$ oraz $f=0$ (patrz rys. 6.15), dopełnienie zmiennej przez $t=0$ oraz $f=1$. Gdy literał nie występuje w implikancie, to odpowiedni bit słowa t i f przyjmuje wartość 0.

Operację pokrywania implikantów realizujemy za pomocą następującej funkcji (użyliśmy syntaktyki języka object pascala):

```
function Pokrywa(K1,K2:tKostka):Boolean;
begin // Czy K1 pokrywa K2?
  Result:=false;
  if ((K1.t or K2.t) = K2.t) and ((K1.f or K2.f)=K2.f) then Result:=true;
end;
```

(Przypomnijmy, że wartość zmiennej Result jest zwracana po wyjściu z funkcji).

Wiemy już, że konsensus może być implikantem (to jest dla nas interesujące) bądź jest pusty albo równy 0 (mało interesujące, dlaczego?). więc podstawową funkcją naszej implementacji jest sprawdzenie warunku istnienia konsensusu dla pary implikantów.

```
function IstniejeKonsensus(K1, K2 : tKostka):Boolean;
begin
    // T, F, D zmienne globalne typu longword
    Result := true;
    T := K1.t or K2.t;
    F := K1.f or K2.f;
    D := T and F;
    if TylkoJedenBitJestJedynka(D) then exit;
    Result := false;
end;
```

Jeżeli funkcja IstniejeKonsensus zwraca wartość TRUE, to wywoływana jest funkcja DajKonsensus wyliczająca konsensus.

```
procedure DajKonsensus(K1, K2 : tKostka; var KK : tKostka);
begin
    // T, F, D zmienne globalne ustalone w IstniejeKonsensus
    KK.t := T and not D;
    KK.f := F and not D;
end;
```

Gdy $D=0$, wtedy konsensus jest pusty (patrz funkcja IstniejeKonsensus). Jeżeli funkcja TylkoJedenBitJestJedynka(D) zwraca TRUE, to istnieje konsensus. Jeżeli liczba jedynek w słowie D jest większa od jeden – konsensus jest równy 0 (podaj sposób najbardziej efektywnej implementacji funkcji TylkoJedenBitJestJedynka).

Szkic procedury IT generującego implikanty proste funkcji przedstawiliśmy poniżej (zestaw iloczynów został zapamiętany w tablicy *kostki*, tablica *pokryte* została zainicjowana wartościami *false*):

```
ii:=2;
while ii <= ileIloczynow do begin
    for kk:=1 to ii-1 do
        if not (pokryte[kk] or pokryte[ii]) then
            if Pokrywa(kostki[ii],kostki[kk])
            then pokryte[kk]:=true
            else if Pokrywa(kostki[kk],kostki[ii])
            then pokryte[ii]:=true
            else if IstniejeKonsensus(kostki[ii],kostki[kk]) then begin
                DajKonsensus(kostki[ii],kostki[kk], temp);
                ll:=0; jestPokrywany:=false;
                while (ll<ileImplikantow) and not jestPokrywany do begin
                    inc(ll);
                    jestPokrywany:=Pokrywa(kostki[ll],temp);
                end;
                if not jestPokrywany then begin
                    inc(ileImplikantow);
                    kostki[ileImplikantow] := temp;
                    pokryte[ileImplikantow] := false;
                end;
            end;
        end;
    end;
    inc(ii);
end;{while}
```

Implikantami prostymi są te pozycje z tablicy *kostki*, dla których w tablicy *pokryte* jest wartość false.

Algorytm IT można ulepszyć, wykluczając sprawdzenie tych par implikantów, dla których na pewno nie istnieje konsensus, a zająć się tylko potencjalnymi kandydatami jego istnienia. Otóż rozważając k -tą zmienną i parę implikantów, konsensus może istnieć, gdy w jednym implikancie zmienna k jest afirmowana a w drugim dopełniona. Zatem, gdybyśmy podzielili listę implikantów na trzy grupy C_k^0 , C_k^1 , C_k^2 ze względu na to, czy kostka zawiera 0, 1 czy 2 na miejscu k , to wystarczyłoby porównywać tylko pary kostek z grup C_k^0 i C_k^1 a obliczone konsensusy zapamiętać w grupie CC. Kostki z CC porównać z kostkami grup C_k^0 , C_k^1 , C_k^2 ,

usuwając z nich wszystkie pokryte kostki. Łącząc wszystkie grupy w jedną i powtarzając powyższe kroki dla następnej zmiennej k otrzymamy listę implikantów prostych.

Przykład

Rozważmy funkcję $f(abcd) = \bar{a}\bar{c}\bar{d} + \bar{a}c\bar{d} + \bar{a}bcd + b\bar{c}d + ab + \bar{a}bcd$. Przez $\text{Kons}(X,Y)$ oznaczaliśmy operację konsensusu.

Dla $k=1$ otrzymamy następujące trzy listy:

$C_1^0 = \{0200, 0210, 0110, 0111\}$, $C_1^1 = \{1022\}$, $C_1^2 = \{2101\}$. Z pierwszej grupy wykreśliliśmy implikant 0110, ponieważ jest pokryty przez 0210.

$\text{Kons}(0200, 1022) = 2000$ $\text{Kons}(0210, 1022) = 2010$ $\text{Kons}(0111, 1022) = 0$

Czyli $CC = \{2000, 2010\}$. Łączymy wszystkie listy wykreślając pokryte implikanty (tutaj nie ma pokrytych implikantów).

Dla $k=2$ otrzymujemy następujące trzy listy:

$C_2^0 = \{1022, 2000, 2010\}$, $C_2^1 = \{2101, 0111\}$, $C_2^2 = \{0200, 0210\}$.

$\text{Kons}(1022, 2101) = 1201$ $\text{Kons}(2000, 2101) = 0$ $\text{Kons}(2010, 2101) = 0$

$\text{Kons}(1022, 0111) = 0$ $\text{Kons}(2000, 0111) = 0$ $\text{Kons}(2010, 0111) = 0$

Czyli $CC = \{1201\}$ i znowu nie ma pokrytych implikantów.

Dla $k=3$ otrzymujemy następujące listy:

$C_3^0 = \{2000, 2101, 0200, 1201\}$, $C_3^1 = \{2101, 0210, 0111\}$, $C_3^2 = \{1022\}$.

$\text{Kons}(2000, 2101) = 2020$ $\text{Kons}(2000, 0210) = 0020$ $\text{Kons}(2000, 0111) = 0$

$\text{Kons}(2101, 2101) = 0$ $\text{Kons}(2101, 0210) = 0$ $\text{Kons}(2101, 0111) = 0121$

$\text{Kons}(0200, 2101) = 0020$ $\text{Kons}(0200, 0210) = 0220$ $\text{Kons}(0200, 0111) = 0$

$\text{Kons}(1201, 2101) = 0$ $\text{Kons}(1201, 0210) = 0$ $\text{Kons}(1201, 0111) = 0$

Czyli $CC = \{2020, 0020, 0220, 0121\}$. Łączymy wszystkie listy i wykreślamy pokryte implikanty: $\{2000, 2101, 0200, 1201, 2010, 0210, 1022, 2020, 0020, 0220, 0121\}$. Wszystkie wykreślone implikanty pokryte są przez 0220 albo 2020 (proszę wskazać odpowiednie pary).

Dla $k=4$ otrzymujemy następujące listy:

$C_4^0 = \{2020, 0220\}$, $C_4^1 = \{2101, 1201, 0121\}$, $C_4^2 = \{1022\}$.

$\text{Kons}(2020, 2101) = 0$ $\text{Kons}(2020, 1201) = 1002$ $\text{Kons}(2020, 0121) = 0$

$\text{Kons}(0220, 2101) = 0102$ $\text{Kons}(0220, 1201) = 0$ $\text{Kons}(0220, 0121) = 0122$

Lista $CC = \{0102, 1002, 0122\}$. Łączymy wszystkie listy i wykreślamy implikanty pokryte:

$\{2020, 0220, 2101, 1201, 0121, 1022, 0122, 0102, 1002\}$.

Wyczerpaliśmy wszystkie zmienne, więc otrzymany zbiór jest zbiorem implikantów prostych: $\bar{b}\bar{d}$, $\bar{a}\bar{d}$, $b\bar{c}d$, $a\bar{c}d$, ab , $\bar{a}b$.

Minimalne pokrycie funkcji możemy uzyskać wykorzystując tablicę implikantów, chociaż możemy także wykorzystać iteracyjny konsensus. Wystarczy zauważyć, że jeżeli z listy wszystkich implikantów WI, tymczasowo usuniemy jeden z nich, np. α i dla tak okrojonej listy zastosujemy algorytm IT, to jeżeli α nie jest zasadniczy, wtedy wszystkie iloczyny zupełne, które on pokrywa są także pokrywane przez inne implikanty pozostałe na liście, czyli IT wygeneruje ponownie implikant α . Jednak, gdy go nie wygeneruje, to α jest zasadniczy.

Stosując powyższe dla wszystkich implikantów prostych z listy WI otrzymamy listę implikantów zasadniczych, którą oznaczmy przez Z.

Zastosujmy algorytm IT do listy Z. W wyniku wygenerujemy pewne iloczyny, które nie muszą być implikantami prostymi, ale jeżeli niektóre z nich są implikantami prostymi znajdującymi się na liście WI, to są one nadmiarowe. Usuwamy z listy WI implikanty znajdujące się na liście Z oraz wszystkie nadmiarowe.

Jeżeli lista WI jest pusta, to minimalne pokrycie zawiera tylko implikanty zasadnicze.

Gdy lista WI nie jest pusta, to należy przystąpić do poszukiwania pseudozasadniczych implikantów (patrz omówienie tablicy implikantów prostych). Dla każdego implikantu prostego α pozostającego w WI, powinniśmy dowiedzieć się czy α i implikanty zasadnicze z Z pokrywają pewien implikant β z WI, ponieważ każdy taki β może być usunięty z listy. Aby się o tym przekonać wykorzystamy oczywiście algorytm IT na tymczasowej liście Z plus implikant α . Jeżeli jakkolwiek implikant β zawarty w liście WI zostanie wygenerowany, to może być usunięty z listy WI. Postępując iteracyjnie dla każdego implikantu z listy WI mamy szansę wygenerowania pseudozasadniczych implikantów.

6.4.3 Poszukiwanie minimalnego pokrycia funkcji

Omawiając metodę QM oraz IT omówiliśmy, przekazując Czytelnikowi intuicję i terminologię dotyczącą minimalnego pokrycia. Problem poszukiwania minimalnego pokrycia jest problemem trudnym, szczególnie dla praktycznych funkcji dużej liczby zmiennych.

Wprowadźmy następujące oznaczenia:

- IP – lista wszystkich implikantów prostych funkcji,
- MP – lista minimalnego pokrycia,
- F – lista iloczynów zupełnych, dla których $f=1$ (pomijamy iloczyny obojętne).

Przystępując do poszukiwania pokrycia mamy daną listę IP oraz F, a lista MP jest jeszcze pusta. Na początku przesuwamy z IP do MP wszystkie zasadnicze implikanty proste. Jeżeli takie istnieją, to z listy F usuwamy te wszystkie iloczyny zupełne, które pokrywane są przez przesunięte implikanty zasadnicze. W wyniku lista F może zostać opróżniona albo zawierać pewne iloczyny. Jeżeli jest pusta to algorytm się kończy, ponieważ implikanty znajdujące się w MP stanowią minimalne pokrycie funkcji.

Gdy lista F nie jest pusta, czyli pozostały na niej te jedynki (nazwiemy je zaległymi), które nie są pokrywane przez implikanty zasadnicze, musimy z pozostałych na liście IP implikantów wybrać taki, który minimalizuje koszt realizacji układowej.

Krok 1. Szukamy implikantów nadmiarowych, czyli takich, że część wspólna listy F i jedynek pokrywanych przez te implikanty jest pusta. Implikanty te możemy usunąć z listy IP.

Krok 2. Szukamy implikantów niesprzyjających minimalizacji. Załóżmy, że implikant prosty p^i pokrywa wszystkie zaległe jedynki, które są także pokrywane przez implikant p^j a koszt realizacji p^i jest nie mniejszy niż realizacji p^j , to p^i może być usunięty z IP. Warunek ten zazwyczaj zapisujemy wykorzystując operacje wyostrzania #:

- (i) $(p^i \cap F) \# p^j = \emptyset$,
- (ii) $\text{koszt}(p^i) \geq \text{koszt}(p^j)$

gdzie $A \# B$, to wszystkie iloczyny zupełne, które należą do A ale nie należą do B.

Krok 3. Szukamy implikantów pseudozasadniczych. Usunięcie z listy IP implikantów nadmiarowych i niesprzyjających spowodowało, że pewne jedynki zaległe są niepokryte

i jest szansa wskazania implikantu, który na tym etapie jest zasadniczym (dlatego nazywamy je też pseudozasadniczymi). Można do tego wykorzystać iteracyjny konsensus. Implikant zasadniczy przesuwamy do listy MP, redukujemy listę F o jedynki pokrywane przez ten implikant i przechodzimy do kroku 1.

Algorytm kończy się jeżeli:

- a) lista F jest pusta. Minimalne pokrycie jest na liście MP.
- b) lista F nie jest pusta, lecz nie możemy znaleźć żadnego implikantu niesprzającego minimalizowaniu kosztu. Mówimy, że napotkaliśmy na cykliczny rdzeń (*cyclic core*) struktury implikantów, który musimy przerwać. Najprościej postępując, wybieramy z listy IP ten implikant, który ma minimalny koszt i przenosimy go do listy MP, usuwamy z listy F iloczyny pokrywane przez przesunięty implikant i cały proces powtarzamy od kroku 1.

Przykład

Rozważmy funkcję $f(abcd) = \Sigma(2,3,6,8,9,10,11,12,14) + \Sigma_{\emptyset}(1,7)$, dla której wszystkie implikanty proste w notacji pozycyjnej to $IP = \{1:0212, 2:2021, 3:2012, 4:1022, 5:2210, 6:1220\}$. Przed dwukropkiem wpisaliśmy arbitralnie dobrany numer implikantu. Lista $F = \{0010, 0011, 0110, 1000, 1001, 1010, 1011, 1100, 1110\}$. Zwróćmy uwagę, że na tej liście nie ma iloczynów obojętnych, dlatego jedynym zasadniczym implikantem prostym dla tej listy jest 6:1220, który przesuniemy do listy MP, opróżnimy listę F z iloczynów pokrywanych przez ten implikant otrzymując:

$$\begin{aligned} MP &= \{6:1220\} \\ IP &= \{1:0212, 2:2021, 3:2012, 4:1022, 5:2210\} \\ F &= \{0010, 0011, 0110, 1001, 1011\} \end{aligned}$$

Problem sprowadza się teraz do takiego wyboru (niezasadniczych) implikantów z listy IP i przesuwaniu ich na listę MP, które minimalizują kryterium kosztu. Sposób ten opisaliśmy wyżej.

Krok 1. Eliminujemy z listy IP implikanty nadmiarowe, czyli takie, które nie mają części wspólnej z zaległymi iloczynami listy F. Żaden z pozostałych pięciu implikantów prostych nie generuje pustego przecięcia z listą F, dlatego nie ma implikanta nadmiarowego.

Krok 2. Należy zatem poszukać implikantów niesprzających:

$$\begin{aligned} (p^1 \cap F) \# p^2 &= \{0210\} & (p^1 \cap F) \# p^3 &= \{0110\} \\ (p^1 \cap F) \# p^4 &= \{0012, 0210\} & (p^1 \cap F) \# p^5 &= \{0011\} \end{aligned}$$

Widzimy, że implikant p^1 nie może być usunięty. Rozważmy implikant p^2 .

$$\begin{aligned} (p^2 \cap F) \# p^1 &= \{1021\} & (p^2 \cap F) \# p^3 &= \{1001\} \\ (p^2 \cap F) \# p^4 &= \{0011\} & (p^2 \cap F) \# p^5 &= \{2011, 1021\} \end{aligned}$$

Implikant p^2 nie może być usunięty. Rozważmy p^3 .

$$\begin{aligned} (p^3 \cap F) \# p^1 &= \{1011\} & (p^3 \cap F) \# p^2 &= \{0010\} \\ (p^3 \cap F) \# p^4 &= \{0012\} & (p^3 \cap F) \# p^5 &= \{2011\} \end{aligned}$$

Implikant p^3 nie może być usunięty. Rozważmy p^4 .

$$\begin{aligned} (p^4 \cap F) \# p^1 &= \{1021\} \\ (p^4 \cap F) \# p^2 &= \emptyset \end{aligned}$$

Ostatnia linijka mówi, że zaległe jedynki F pokrywane przez p^4 są także pokrywane przez p^2 . Ponieważ $\text{Koszt}(p^4) \geq \text{Koszt}(p^2)$ więc p^4 możemy usunąć z listy. Czyli $IP = \{1:0212, 2:2021,$

3:2012, 5:2210}.

Krok 3. Na tym etapie jedynym pseudozasadniczym implikantem jest p^2 , który jako jedyny pokrywa 1001. Krok 2 kończy się następującym wynikiem:

$$IP=\{1:0212, 3:2012, 5:2210\}, MP=\{6:1220, 2:2021\}, F=\{0210\}$$

Przechodzimy do kroku 1.

Krok 1'. Zauważamy znowu, że koszt realizacji każdego implikantu z IP jest identyczny oraz nie ma implikantów nadmiarowych.

Krok 2'. Przystępujemy do szukania implikantów niesprzyjających minimalizacji kosztu.

$$(p^1 \cap F) \# p^3 = \{0110\}$$

$$(p^1 \cap F) \# p^5 = \emptyset \text{ oraz } \text{koszt}(p^1) \geq \text{koszt}(p^5)$$

czyli p^1 usuwamy z listy IP otrzymując {3:2012, 5:2210}.

Krok 3'. Zasadniczym implikantem jest {5:2210}, bo tylko on pokrywa zaległą jedynkę 0110.

Czyli otrzymujemy: $IP=\{3:2012\}$, $MP=\{6:1220, 2:2021, 5:2210\}$ oraz $F=\emptyset$. Lista F jest pusta więc MP zawiera minimalne pokrycie.

Przykład

Rozważmy funkcję $f(abcd)=\Sigma(2,3,5,7,8,10,12,13)$. Wszystkie implikanty proste, które otrzymaliśmy stosując iteracyjny konsensus są następujące: $IP=\{1:0012, 2:0211, 3:0121, 4:2010, 5:1020, 6:1200, 7:2101, 8:1102\}$. Podobnie jak w poprzednim przykładzie, przed dwukropkiem wpisaliśmy arbitralnie dobrany numer kostki. Zastosowanie iteracyjnego konsensusu do wygenerowania implikantów zasadniczych daje zbiór pusty (brak jest implikantów zasadniczych), czyli $MP=\emptyset$, oraz $F=\{0010, 0011, 0101, 0111, 1000, 1010, 1100, 1101\}$. Nie ma implikantów nadmiarowych.

Spróbujmy wskazać implikanty niesprzyjające, wyliczając $(p^i \cap F) \# p^j$. Okazuje się, że warunek dla dowolnej pary implikantów i listy F nie daje listy pustej. Czyli istnieje cykliczny rdzeń. W takiej sytuacji arbitralnie przesuwamy na listę MP implikant $p^1 = 0012$. Wtedy zaległe jedynki tworzą listę $F=\{0101, 0111, 1000, 1010, 1100, 1101\}$, dla której $IP=\{3:0121, 5:1020, 6:1200, 7:2101, 8:1102\}$. Na tym etapie na liście IP mamy dwa implikanty pseudozasadnicze {3:0121, 5:1020}, które przenosimy do MP, a zredukowana lista $F=\{1100, 1101\}$ jest całkowicie pokrywana przez implikant {8:1102}, czyli:

$$IP=\emptyset, \quad MP=\{1:0012, 3:0121, 5:1020, 8:1102\}, \quad F=\emptyset.$$

Kończymy wykonanie algorytmu, ponieważ lista F jest pusta, czyli na liście MP są implikanty minimalnego pokrycia.

Jeżeli algorytm rozpoczniemy od innego implikantu niż p^1 to otrzymamy inne minimalne pokrycie. Dla przykładu, gdy wybierzemy jako pierwszy implikant p^2 to otrzymamy: {2:0211, 4:2010, 6:1200, 7:2101 }.

6.4.4 Metoda Espresso

Nazwa metody pochodzi od programu Espresso, opracowanego i intensywnie rozwijanego w latach 80-tych na uniwersytecie kalifornijskim w Berkeley. W niniejszym rozdziale postaramy się przedstawić najważniejsze z funkcji metody, których wykorzystanie prowadzi do otrzymania pokrycia prostego, nienadmiarowego (także minimalnego). Podstawowymi proce-

durami są: procedura rozwijania (EXPAND), zwijania (REDUCE), usuwania nadmiarowości (IRREDUNDANT COVER) oraz dopełniania zadanego pokrycia (COMPLEMENT) czy wyznaczania implikantów zasadniczych (ESSENTIALS).

Podstawową cechą odróżniającą metodę Espresso od metody QM, jest to, że nie próbuje ona generować wszystkich implikantów prostych - unikając tym samym ewentualnych trudności z ich zapamiętaniem - lecz stara się tak modyfikować zadane pokrycie funkcji, by pokrycie wynikowe spełniało zadane warunki kosztu.

Procedura EXPAND zmienia pokrycie funkcji na takie, w którym każdy implikant jest prosty, usuwając te implikanty, które są pokrywane przez nowe implikanty proste.

Procedura IRREDUNDANT odpowiada za zmniejszenie liczby implikantów wchodzących w skład pokrycia (oczywiście ta zmniejszona liczba implikantów nadal pokrywa wszystkie jedynki funkcji).

Procedura ESSENTIALS wyznacza implikanty zasadnicze funkcji, ponieważ wiemy już, że one muszą wejść do pokrycia.

Procedura REDUCE zmniejsza rozmiary implikantów, ale w taki sposób, aby każda jedynka zadanej funkcji nadal była pokryta przez co najmniej jeden implikant.

```
ESPRESSO(ON,DC)
OFF:=Complement(ON  $\cup$  DC)
F:=Expand(ON,OFF)
F:=Irredundant(F,DC)
E:=Essentials(F,DC)
F:=F - E
DC=DC  $\cup$  E
REPEAT
  C2= Koszt(F)
  REPEAT
    C1=|F|
    F:=Reduce(F,DC)
    F:=Expand(F,OFF)
    F:=Irredundant(F,DC)
  UNTIL (|F|<C1)
  F:=Last_Gasp(F,DC,OFF)
UNTIL(Koszt(F) < C2)
F:=F  $\cup$  E
DC:=DC-E
F:=Make_Sparse(F,DC,OFF)
```

Rys. 6.16 Pętla główna programu Espresso

Główna pętla programu Espresso, w skład której wchodzi kolejne wywołania procedur, powtarzana jest tak długo, jak długo kolejna iteracja powoduje ulepszenie istniejącego pokrycia. Innymi słowy, jeżeli kolejna iteracja nie zmniejszyła liczby implikantów, następuje wyjście z pętli (Rys. 6.16). Na rysunku widzimy jeszcze dwie dodatkowe procedury – Last_Gasp oraz Make_Sparse – których omawianiem nie będziemy się zajmować.

Przed przystąpieniem do omówienia procedur programu Espresso, wprowadzimy nowe pojęcia, które są też cechą charakterystyczną tej metody.

Danymi wejściowymi programu jest zbiór włączenia (ON) i nieokreśloności (DC), czyli odpowiednio zestaw tych kostek (implikantów) funkcji, dla których $f=1$ bądź $f=\emptyset$. Zbiór wyłączenia (OFF) jest wyznaczany za pomocą procedury COMPLEMENT jako dopełnienie sumy zbioru ON i DC (w ogólnym przypadku zadane mogą być dowolne dwa z powyższych trzech zbiorów, ponieważ trzeci zawsze można wyliczyć).

Aby ułatwić wykonywanie operacji na implikantach (kostkach), poszczególne literały implikantów wejściowych są kodowane za pomocą 2-bitowych pól w następujący sposób: zmienną afirmowaną zastępujemy przez 01, zmienną dopełnioną przez 10 a zmienną której brak w implikancie kodujemy 11. Czwarte możliwe kodowanie 00 powstaje zazwyczaj na etapie wykonywania operacji na zakodowanych implikantach i oznacza „pusty” implikant, czyli taki, którym się nie zajmujemy. Wyjaśni się to w dalszej części opisu.

Przy okazji zwróćmy uwagę na to, że sposób zakodowania kostek jest zupełnie odmienny niż w prezentowanej wyżej metodzie iteracyjnego konsensusu.

Założymy ponadto, że interesować nas będzie poszukiwanie pokrycia tylko dla pojedynczych funkcji, czyli kostki nie będą posiadać kodowania wyjścia (wartości funkcji). W ogólnym przypadku metoda Espresso dotyczy minimalizacji zestawu funkcji i wtedy w każdej kostce po 2^n bitach, n – liczba zmiennych, występuje część wyjściowa, na której koduje się wartości zestawu funkcji.

UWAGA: W dalszej części opisu metody Espresso, pod pojęciem kostki (implikantu) będziemy rozumieć tylko i wyłącznie zakodowane kostki (implikanty). W szczególności wszystkie operacje są wykonywane na zakodowanych kostkach. Ma to zachęcić do próby zakodowania podstawowych procedur metody Espresso np. w object pascalu.

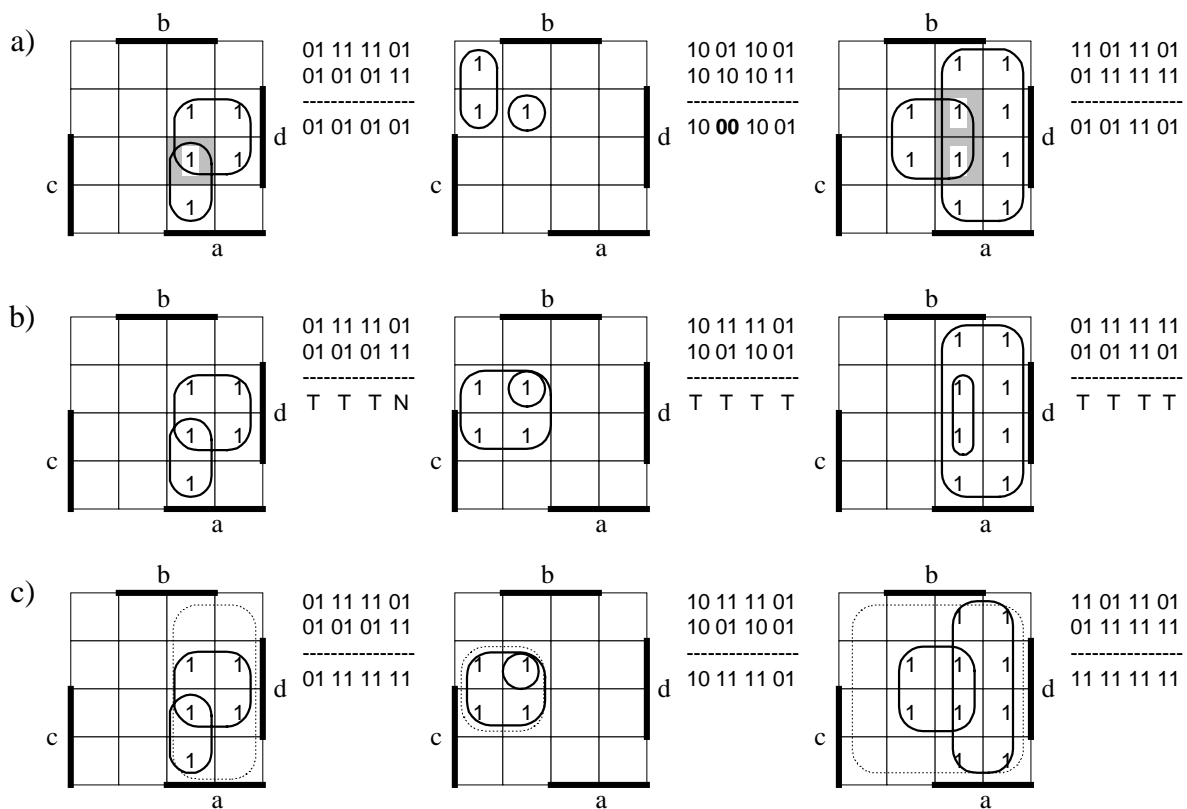
Przykład:

Przykładowe kodowania kostek dla funkcji 4 zmiennych abcd są następujące:

$$\begin{array}{lll} \bar{a}\bar{b}\bar{c} = 10\ 10\ 10\ 11 & abcd = 01\ 01\ 01\ 01 & \bar{a}\bar{b}cd = 10\ 01\ 10\ 01 \\ ac = 10\ 11\ 10\ 11 & \bar{a} = 10\ 11\ 11\ 11 & U = 11\ 11\ 11\ 11 \end{array}$$

Kostka zawierająca same jedynki nazywana jest kostką tautologiczną i oznacza całą przestrzeń danej liczby zmiennych.

Powyższy sposób kodowania implikantów, być może wydłużający jego reprezentację w słowie komputerowym, znakomicie ułatwia wykonywanie podstawowych operacji na implikantach, takich jak obliczenie części wspólnej czy pokrywania się dwóch implikantów, ponieważ są one efektywnie realizowane na każdym komputerze.



Rys. 6.17 Przykłady operacji wykonywanych na zakodowanych kostkach, a) część wspólna dwóch kostek, b) pokrywanie dwóch kostek, c) suma logiczna dwóch kostek (superkostka).

Przykład:

Część wspólna dwóch implikantów jest po prostu iloczynem ich bitów (Rys.6.17a). Gdy w wyniku otrzymamy pole 00 (drugi przykład na rys.6.17a), to rozważane implikanty nie mają części wspólnej. Implikant pokrywa inny, jeśli jego wszystkie bity pokrywają bity drugiego (są większe lub równe). Porównaj rys.6.17b. Suma logiczna bitów dwóch implikantów daje w wyniku najmniejszy implikant je pokrywający (superkostkę, rys.6.17c).

Odległość między dwoma implikantami definiujemy jako liczbę pustych pól 00 w ich części wspólnej. Nietrudno zauważyć, że dla implikantów przecinających się (mających wspólną część) odległość jest równa 0, a dla nie przecinających się odległość jest większa od zera.

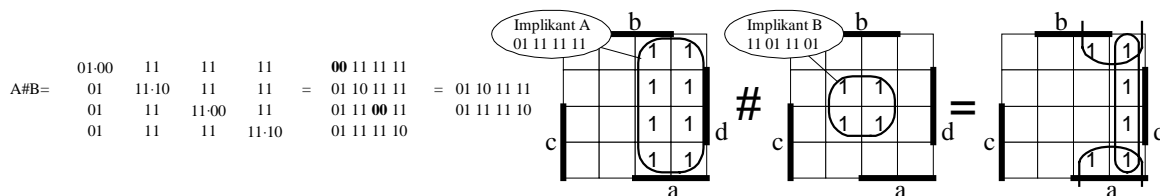
Wykorzystywana wcześniej operacja wyostrzania (#) na dwóch implikantach daje w wyniku zbiór implikantów, które pokrywają wszystkie i tylko te iloczyny zupełne, które są pokryte przez pierwszy implikant i nie są pokryte przez drugi (zatem operacja nie jest przemienne). Jeżeli implikant $A=a_1a_2...a_n$ oraz $B=b_1b_2...b_n$, gdzie a_i oraz b_i są zakodowanymi za pomocą dwóch bitów polami, to operacja wyostrzania realizowana jest następująco:

$$A \# B = \begin{cases} a_1 \cdot \bar{b}_1 & a_2 & a_3 & \dots & a_n \\ a_1 & a_2 \cdot \bar{b}_2 & a_3 & \dots & a_n \\ a_1 & a_2 & a_3 \cdot \bar{b}_3 & \dots & a_n \\ \dots & \dots & \dots & \dots & \dots \\ a_1 & a_2 & a_3 & \dots & a_n \cdot \bar{b}_n \end{cases}$$

gdzie ($a_i \cdot \bar{b}_i$ to część wspólna pola a_i i dopełnienia pola b_i):

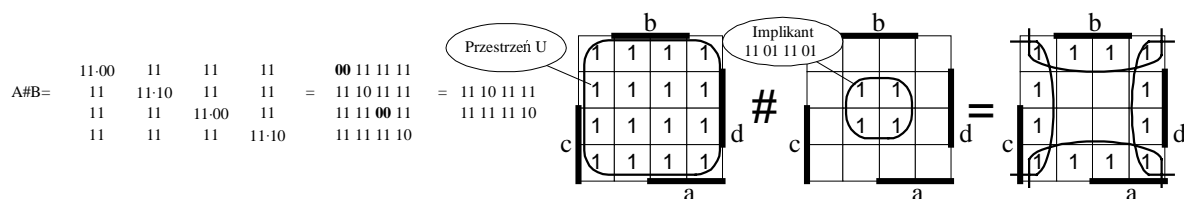
Przykład:

Wykonajmy operację wyostrzania na dwóch implikantach: $A=01\ 11\ 11\ 11$ oraz $B=11\ 01\ 11\ 01$. Wynik operacji przedstawiliśmy na rys. 6.18.



Rys. 6.18 Ilustracja operacji wyostrzania

Operacja wyostrzania może być przydatna do wyznaczenia dopełnienia implikantu. W tym celu wystarczy wykonać wyostrzanie na parze złożonej z całej przestrzeni $U=11\ 11\ \dots\ 11$ oraz danego implikantu (Rys. 6.19).



Rys. 6.19 Dopełnienie implikantu z wykorzystaniem wyostrzania

Procedura COMPLEMENT

Jak wspomnieliśmy powyżej do obliczenia dopełnienia, możemy zastosować operacje wyodręszania. Jednak w metodzie Espresso zastosowano bardziej efektywny algorytm, wykorzystujący następującą zależność:

$$\bar{f} = x \cdot \bar{f}_x + \bar{x} \cdot \bar{f}_{\bar{x}}$$

Zatem, dopełnienie funkcji f można wyznaczyć rekurencyjnie korzystając z dopełnionych kofaktorów funkcji. Ponieważ założyliśmy, że będziemy operować na zbiorach (pokryciach funkcji), dlatego powyższe zależność w stosunku do pokryć możemy przepisać następująco:

$$\bar{F} = (C(x) \cap \bar{F}_x) \cup (C(\bar{x}) \cap \bar{F}_{\bar{x}})$$

gdzie: \bar{F} - zbiór kostek dopełnienia pokrycia F

$C(x), C(\bar{x})$ - kostka odpowiednio literału x i \bar{x} (np. $C(x_2)=11\ 01\ 11\ 11\dots$)

$\bar{F}_x, \bar{F}_{\bar{x}}$ - zbiory kostek stanowiących pokrycia dopełnień odpowiednio dla \bar{f}_x oraz $\bar{f}_{\bar{x}}$

\cap, \cup - przecięcie oraz suma zbiorów

Jeżeli jest spełniony jeden z poniższych warunków to rekurencyjne wywołanie procedury kończymy, ponieważ możemy w prosty sposób obliczyć dopełnienie kofaktora:

1. Pokrycie F jest puste. Dopełnieniem jest cała przestrzeń.
2. Pokrycie F zawiera wiersz samych jedynek. Dopełnienie jest zbiorem pustym.
3. Pokrycie F zawiera jedną kostkę. Dopełnienie wyliczamy z prawa De Morgana.
4. Wszystkie implikanty pokrycia F zależą od jednej zmiennej i nie ma kolumny zawierającej same zera. Dopełnienie jest zbiorem pustym.
5. Pewna i -ta kolumna pokrycia zawiera same zera i pewna kostka α zawiera same jedynki oprócz i -tej kolumny. Dopełnienie jest równe $\bar{F}_\alpha \cup \bar{\alpha}$.

Kofaktor implikantu α ze względu na implikant β jest pusty, gdy α nie przecina β . W przeciwnym przypadku jest ono wyliczane w następujący sposób:

$$\alpha_\beta = a_1 + \bar{b}_1 \ a_2 + \bar{b}_2 \ \dots \ a_n + \bar{b}_n$$

Analogicznie kofaktor zbioru implikantów ze względu na implikant jest zbiorem odpowiednich kofaktorów (czyli to co nas będzie interesować).

Przykład:

Wyliczmy kofaktory dla pokrycia

$$F = \left\{ \begin{array}{l} \alpha \ 10 \ 11 \ 10 \ 11 \\ \beta \ 01 \ 11 \ 11 \ 01 \\ \chi \ 11 \ 11 \ 01 \ 10 \end{array} \right\}$$

względem zmiennej a , czyli kostki $C(a) = 01 \ 11 \ 11 \ 11$. Kostka α nie przecina $C(a)$, czyli kofaktor α względem kostki $C(a)$ jest pusty. Zatem zgodnie z powyższą zależnością mamy:

$$F_a = \left\{ \begin{array}{l} 01 \ 11 \ 11 \ 01 \\ 11 \ 11 \ 01 \ 10 \end{array} \right\}_{01 \ 11 \ 11 \ 11} = \left\{ \begin{array}{l} 01+10 \ 11+00 \ 11+00 \ 01+00 \\ 11+10 \ 11+00 \ 01+00 \ 10+00 \end{array} \right\} = \left\{ \begin{array}{l} 11 \ 11 \ 11 \ 01 \\ 11 \ 11 \ 01 \ 10 \end{array} \right\}$$

Analogicznie wyliczamy

$$F_{\bar{a}} = \left\{ \begin{matrix} 10 & 11 & 10 & 11 \\ 11 & 11 & 01 & 10 \end{matrix} \right\}_{10 \ 11 \ 11 \ 11} = \left\{ \begin{matrix} 10+01 & 11+00 & 10+00 & 11+00 \\ 11+01 & 11+00 & 01+00 & 10+00 \end{matrix} \right\} = \left\{ \begin{matrix} 11 & 11 & 10 & 11 \\ 11 & 11 & 01 & 10 \end{matrix} \right\}$$

Oczywiście $F = (C(a) \cap F_a) \cup (C(\bar{a}) \cap F_{\bar{a}})$ co łatwo sprawdzić:

$$(01 \ 11 \ 11 \ 11 \cap \left\{ \begin{matrix} 11 & 11 & 11 & 01 \\ 11 & 11 & 01 & 10 \end{matrix} \right\}) \cup (10 \ 11 \ 11 \ 11 \cap \left\{ \begin{matrix} 11 & 11 & 10 & 11 \\ 11 & 11 & 01 & 10 \end{matrix} \right\}) =$$

$$\left\{ \begin{matrix} 01 & 11 & 11 & 01 \\ 01 & 11 & 01 & 10 \end{matrix} \right\} \cup \left\{ \begin{matrix} 10 & 11 & 10 & 11 \\ 10 & 11 & 01 & 10 \end{matrix} \right\} = \left\{ \begin{matrix} 01 & 11 & 11 & 01 \\ 01 & 11 & 01 & 10 \\ 10 & 11 & 10 & 11 \\ 10 & 11 & 01 & 10 \end{matrix} \right\} = \left\{ \begin{matrix} \alpha & 10111011 \\ \beta & 01111101 \\ \chi & 11110110 \end{matrix} \right\}$$

Zatem samo wyliczenie kofaktorów nie sprawia trudności. Jednak nam zależy na wyliczeniu dopełnień kofaktorów.

Zanim do tego przejdziemy podamy heurystyczne metody wyboru zmiennej, względem której dokonujemy rozkładu funkcji (okrycia). Problem wyboru zmiennej można by zignorować (tzn. rozwijać pokrycie według kolejnych zmiennych np. a,b,c,...) jednak okazuje się, że odpowiednia kolejność wyboru zmiennych do rozwijania ma znaczący wpływ na szybkość dochodzenia do warunków kończących wywołanie rekurencyjne.

W metodzie Espresso zaproponowano następującą strategię wyboru zmiennej:

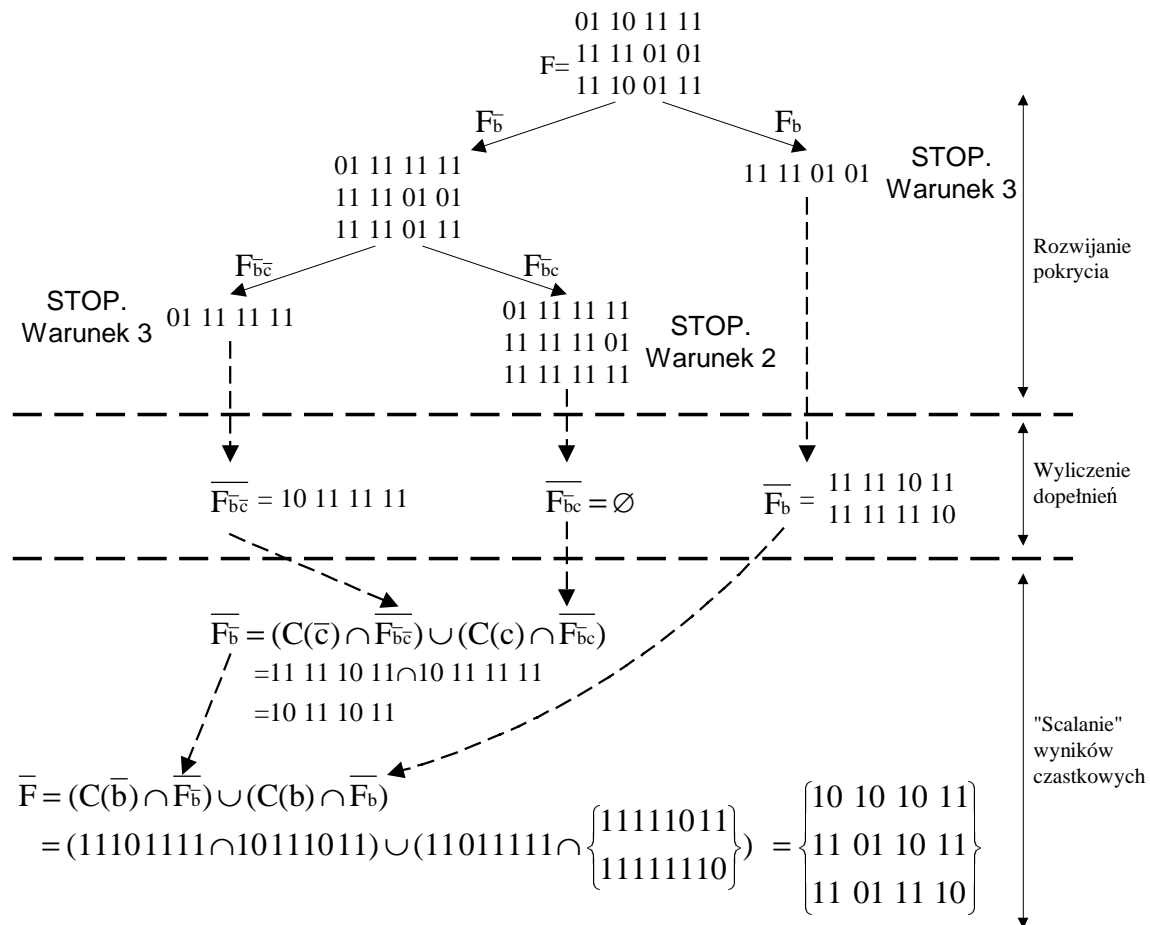
1. Wybieramy zmienną, której pola w kostce mają różne polaryzacje i występuje w największej liczbie kostek. Różna polaryzacja zmiennej oznacza, że we wszystkich polach odpowiadających tej zmiennej jest przynajmniej jedno pole 01 oraz przynajmniej jedno pole 10. O takiej zmiennej mówimy, że jest niejednorodna.
2. Jeżeli mamy więcej niż jedną taką zmienną, to możemy wyliczyć moduł różnicy liczby polaryzacji i wybrać najmniejszy z nich.

Jeżeli w danym pokryciu nie występuje ani jedna zmienna niejednorodna, wtedy wyboru zmiennej do rozwijania dokonujemy zgodnie z następującymi regułami:

- a) Wybieramy kostki z maksymalną liczbą pól 11,
- b) W kostkach wybieramy zmienne, które mają w odpowiednich polach największą liczbę zer,
- c) Spośród wybranych zmiennych w punkcie 2 wybieramy te, które mają najwięcej zer w odpowiadających polach.

Przykład:

Niech będzie dane pokrycie $F = \{11 \ 01 \ 10 \ 11, 01 \ 10 \ 11 \ 11, 10 \ 11 \ 01 \ 11\}$. Zauważamy, że istnieją trzy równoprawne zmienne niejednorodne (warunek 1 i 2 powyżej). Zaczniemy rozwijać pokrycie od zmiennej a. Kolejne kroki wyliczania kofaktorów oraz ich dopełnień (rekurencyjne rozwijanie pokryć) a także „zwijanie” wyników cząstkowych pokazaliśmy na rys. 6.20. Zachęcamy Czytelników do zweryfikowania otrzymanego wyniku poprzez wpisanie do dwóch tablic Karnaugh’a pokrycia F oraz \bar{F} , a także jako ćwiczenie wyliczenie dopełnienia dla pokrycia $FF = \{11 \ 01 \ 10 \ 11, 01 \ 10 \ 11 \ 11, 10 \ 11 \ 01 \ 11, 11 \ 10 \ 01 \ 01\}$, które reprezentuje tę samą funkcję co pokrycie F.



Rys. 6.20 Ilustracja procedury COMPLEMENT

Procedura EXPAND

Rozwijanie kostek wykonywane jest poprzez zamianę niektórych zer na jedynki w jego zakodowanej reprezentacji. Nietrudno zauważyć, że przesłanką rozwijania jest takie powiększenie kostki, aby pokryć kostki o mniejszych rozmiarach (pokryte kostki będziemy usuwać), ponieważ każda zamiana 0 na 1 dwukrotnie zwiększa rozmiar kostki. Dla przykładu, jeśli w tablicy Karnaugh'a kostka pokrywała dwie kratki, to po zamianie jednego 0 na 1, będzie pokrywać już cztery kratki.

Jedynym ograniczeniem procesu rozwijania kostek jest zbiór OFF, ponieważ powiększona kostka nie może się z nim przeciąć (wtedy rozwinięta kostka nie będzie implikantem). Zatem po każdej zamianie zera na jedynkę należy sprawdzić czy powiększona kostka ma część wspólną ze zbiorem OFF. Gdyby tak się stało, należy zrezygnować z wprowadzania zamiany 0 na 1 na rozważanej pozycji (z powyższego wynika też, że jeżeli wiemy że kostka reprezentuje implikant prosty, to nie ma sensu jej rozwijać – dlaczego?). Wymagany zbiór OFF można wyznaczyć wykorzystując procedurę COMPLEMENT.

Danymi wejściowymi procedury EXPAND jest lista implikantów pokrywających funkcję. W wyniku otrzymujemy listę implikantów prostych (być może nie wszystkich), ale takich, że żaden z nich nie zawiera się w innym (jednak ich suma może pokrywać pewien wybrany implikant).

Rezultaty rozwijania kostek silnie zależą zarówno od:

- a) kolejności wyboru kostek,
- b) kolejności zamian zer na jedynki („kierunków” rozwijania).

W obu zagadnieniach stosowane są reguły heurystyczne omówione poniżej.

Ustalanie kolejności wyboru kostek do rozwijania.

Aby dokonać wyboru kostki do rozwijania, należy każdej z nich przyporządkować pewną wartość nazywaną wagą. Sposób wyliczania wag nie jest skomplikowany i zostanie przedstawiony dla funkcji $f(abc)=\Sigma(0,1,2,7)+\Sigma_{\emptyset}(3)$, czyli zbioru $ON=\{\alpha=10\ 01\ 10, \beta=10\ 10\ 01, \gamma=10\ 10\ 10, \delta=01\ 01\ 01\}$, zbioru $DC=\{10\ 01\ 01\}$ oraz zbioru $OFF=\{01\ 01\ 10, 01\ 10\ 01, 01\ 10\ 10\}$. Poszczególne kostki zbioru ON nazwaliśmy dodatkowo literkami alfabetu greckiego.

Na początku wyliczamy wektor sum kolumnowych dla zbioru ON (Rys.6.21a), licząc jedynki w każdej kolumnie, a następnie wektor wag (Rys.6.21b), który otrzymujemy jako proste arytmetyczne wymnożenie macierzy ON i transpozycji wektora sum kolumnowych.

Zatem kostki α , β i γ mają wagę równą 7, a kostka δ ma wagę 5.

a)

α	1	0	0	1	1	0
β	1	0	1	0	0	1
γ	1	0	1	0	1	0
δ	0	1	0	1	0	1
	3	1	2	2	2	2

b)

1	0	0	1	1	0
1	0	1	0	0	1
1	0	1	0	1	0
0	1	0	1	0	1

 \cdot

3
1
2
2
2
2

 $=$

7
7
7
5

Rys. 6.21 Wyliczanie wartości a) sum kolumnowych, b) wartości wag kostek

Wagi przypisane każdej kostce wskazują jak bardzo się ona różni od pozostałych kostek z pokrycia. Im waga mniejsza, tym jest mniej literałów różniących dany implikant od pozostałych. Zatem implikanty o mniejszych wagach mają mniejsze prawdopodobieństwo być pokrytym przez inne implikanty (mają one mało jedynek, które znajdują się w kolumnach o dużej liczbie jedynek). Dlatego sortujemy kostki według malejących wartości wag i zaczynamy rozwijanie od tych posiadających **mniejsze** wagi.

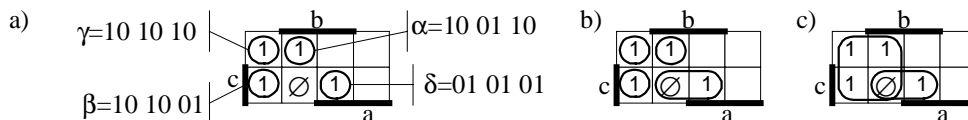
Proces rozwijania zilustrujemy za pomocą przykładu.

Przykład:

Nie będzie dana funkcja z rys.6.21a. Rozwijanie zaczniemy od kostki δ , ponieważ ma najmniejszą wagę równą 5. Ponieważ jeszcze nie omówiliśmy zasad ustalania kolejności wyboru zmian zer na jedynki (zrobimy to w następnym punkcie), dlatego wyłącznie w niniejszym przykładzie zaczniemy rozwijanie w kolejności od lewej do prawej. Jeżeli w kostce $\delta=01\ 01\ 01$ zamienimy pierwsze z lewej strony zero na jedynkę, to otrzymamy kostkę $11\ 01\ 01$. Nie przecina się ona ze zbiorem OFF, dlatego nowa kostka jest nadal implikantem funkcji. Zamiana na jedynkę kolejnego zera daje nam kostkę $11\ 11\ 01$, która przecina się ze zbiorem OFF, dlatego tej zamiany nie możemy dokonać (kostka nie implikuje funkcji). Podobnie zamiana zera znajdującego się na piątej pozycji nie jest dozwolona. Zatem „najlepiej” rozwinięta kostka ma postać $\delta\delta=11\ 01\ 01$, nie pokrywa żadnej innej kostki oraz kostka ta

reprezentuje implikant prosty (Rys.6.22b). W wyniku, do nowego pokrycia naszej funkcji wchodzi następujące kostki: $\{\delta=11\ 01\ 01, \alpha=10\ 01\ 10, \beta=10\ 10\ 01, \gamma=10\ 10\ 10\}$.

Przystępujemy do rozwijania kolejnych kostek. Ponieważ pozostałe kostki (tzn. α , β oraz γ) mają identyczne wagi równe 7, dlatego arbitralnie wybierzemy kostkę $\alpha=10\ 01\ 10$. Zamiana pierwszego zera nie jest dozwolona, możemy natomiast zmienić zero na trzeciej pozycji. Po wykonaniu tej zamiany dozwolona jest również zamiana na pozycji szóstej. W rezultacie otrzymujemy kostkę $\alpha\alpha=10\ 11\ 11$ (rys.6.22c). Kostka $\alpha\alpha$ pokrywa kostki β oraz γ dlatego możemy je usunąć bez szkody dla pokrycia, które teraz jest następujące: $\{\delta=11\ 01\ 01, \alpha\alpha=10\ 11\ 11\}$.



Rys. 6.22 Ilustracja procedury EXPAND, a) zadana funkcja, b) wynik rozwinięcia kostki δ , c) wynik rozwinięcia kostki α

Ustalanie kolejności zamian zer na jedynki.

Wprowadźmy kilka nowych pojęć. Niech α będzie rozwijaną kostką, a ROZWIN zbiorem numerów kolumn tej kostki, w których są zera. W trakcie zamiany 0 na 1 odpowiedni numer kolumny jest usuwany ze zbioru ROZWIN. Rozwijanie kostki się kończy, gdy zbiór ROZWIN jest pusty. Kostką nadrozwinętą nazwiemy kostkę, która powstała przez zamianę 0 na 1 dla kolumn znajdujących się w zbiorze ROZWIN.

Rozwijamy kostkę wykonując następujące kroki:

Krok 1 Tworzymy zbiór ROZWIN dla kostki α .

Krok 2 Ustalamy, które zera rozwijanej kostki nigdy nie mogą być zamienione na jedynki, bo nadrozwinęta kostka przestaje być implikantem. Aby się o tym przekonać poszukujemy dowolnej kostki ze zbioru OFF, której odległość od rozwijanej kostki wynosi 1. Puste pole w części wspólnej tych dwóch kostek wskaże nam element, który usuniemy ze zbioru ROZWIN (odległość powinna być równa 1, bo jeżeli zamienilibyśmy 0 na 1 w kolumnie odpowiadającej polu pustemu, to spowodowałoby zmniejszenie odległości do zera, czyli przecinanie się kostki ze zbiorem OFF).

Krok 3 Poszukujemy kolumn, które zawsze mogą być zamienione z 0 na 1, dokonujemy zamiany i usuwamy odpowiednie elementy ze zbioru ROZWIN. Poszukiwanie to sprowadza się do znalezienia kolumn zbioru OFF zawierających same zera, ponieważ w tych „kierunkach” zawsze możemy rozwinąć kostkę.

Krok 4 Sprawdzamy, czy superkostka rozwijanego implikantu α oraz dowolnego innego implikantu β jest dopuszczalna, czyli jest implikantem funkcji. Jeśli tak, to zamieniamy z 0 na 1 element różniący implikant α oraz β i usuwamy go ze zbioru ROZWIN, z pokrycia usuwamy implikant β i przechodzimy do kroku 2. W przeciwnym przypadku idziemy dalej.

Krok 5 Na koniec poszukujemy największego implikantu prostego. Poszukiwanie to polega na znalezieniu największego podzbioru zbioru ROZWIN, którego elementy można zamienić z 0 na 1 i takiego, by rozwinęty implikant nie miał części wspólnej ze zbiorem OFF.

Przykład:

Niech będzie dana funkcja, dla której zbiór $ON = \{\alpha = 01\ 01\ 10, \beta = 10\ 01\ 01, \gamma = 01\ 10\ 10\}$ a zbiór $OFF = \{\varepsilon = 10\ 01\ 10, \delta = 10\ 10\ 10, \lambda = 10\ 10\ 01\}$. Wagi implikantów są następujące: $\alpha = 6$, $\beta = 4$ oraz $\gamma = 5$.

Zatem rozpoczynamy rozwijanie od implikantu β , dla którego $ROZWIN = \{2, 3, 5\}$. W kroku 2 liczymy odległości od poszczególnych kostek zbioru OFF .

$\beta = 10\ 01\ 01$	$\beta = 10\ 01\ 01$	$\beta = 10\ 01\ 01$
$\varepsilon = 10\ 01\ 10$	$\delta = 10\ 10\ 10$	$\lambda = 10\ 10\ 01$
-----	-----	-----
10 01 00	10 00 00	10 00 01

Jak widać, odległość między β a ε oraz β a λ wynosi 1, a między β a δ równa się 2. Częścią wspólną β i ε jest kostka 10 01 00. Pustym polem jest pole 3, więc ze zbioru $ROZWIN$ usuwamy element 5. W części wspólnej β i λ pustym polem jest pole 2, więc ze zbioru $ROZWIN$ usuwamy element 3. W wyniku $ROZWIN = \{2\}$.

W kroku 3 stwierdzamy, że element 2 w rozwijanej kostce możemy zawsze zamienić na 1, ponieważ druga kolumna zbioru OFF ma same zera. Otrzymujemy nadrozwinętą kostkę $\beta\beta = 11\ 01\ 01$. Usuwamy ze zbioru $ROZWIN$ element 2. Zbiór $ROZWIN$ jest pusty, dlatego kończymy rozwijanie implikantu β .

Jako drugi będzie rozwijany implikant λ , dla którego $ROZWIN = \{1, 4, 6\}$. W kroku 2 usuwamy ze zbioru $ROZWIN$ element 1, bo po rozwinęciu pierwszego zera nadrozwinęta kostka przecina zbiór OFF . Zatem $ROZWIN = \{4, 6\}$. Krok 3 nie wnosi nic nowego, zaś w kroku 4 przekonujemy się, że superkostka(λ, α) jest kostką dopuszczalną, więc zamieniamy 0 na 1 na pozycji 4, usuwamy 4 ze zbioru $ROZWIN$, pozostawiając w nim jeszcze element 6. Usuwamy z pokrycia implikant α . Po powrocie do kroków 2 i 3 przekonujemy się, że one nic nie wnoszą. Podobnie krok 4. W kroku 5 zamieniamy 0 na 1 w kolumnie 6 i usuwamy element ze zbioru $ROZWIN$. Zbiór $ROZWIN$ jest pusty, zatem kończymy rozwijanie implikantu λ , otrzymując $\lambda' = 01\ 11\ 11$.

Ponieważ nie mamy już co rozwijać, w wyniku otrzymaliśmy następujące pokrycie: $\{11\ 01\ 01, 01\ 11\ 11\}$.

Procedura REDUCE.

Zwijanie kostek jest operacją w pewnym sensie odwrotną do rozwijania, ponieważ zmniejszamy rozmiar implikantu (zmniejszamy obwódkę w tablicy Karnaugh'a). Wydawałoby się, że taka operacja nie ma sensu – przecież dążymy do tego by uzyskać implikanty proste, czyli implikanty o maksymalnych rozmiarach. Jednak to świadome „cofnięcie” się do innego punktu startowego, stwarza szansę uzyskania innego pokrycia w kolejnym zastosowaniu operacji rozwijania (być może z mniejszą liczbą implikantów prostych). Takie działanie jest charakterystyczne dla procedur heurystycznych, które iteracyjnie poprawiają wynik.

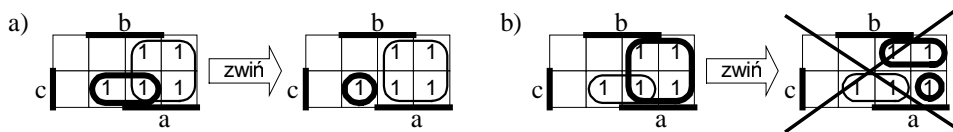
Na operacje zwijania narzucone są pewne ograniczenia:

- zwinięty implikant musi nadal, wraz z pozostałymi implikantami pokrywać funkcję (jednak pokrycie nie musi być proste) oraz
- liczba implikantów musi pozostać bez zmian (niezmieniona moc pokrycia).

Wydaje się, że jest to operacja prosta, ponieważ w celu zwinięcia implikantu α należącego do pokrycia F należy pozbyć się z niego wszystkich iloczynów zupełnych pokrywanych przez $F \setminus \{\alpha\}$. Musimy jedynie zagwarantować, by w wyniku otrzymać jeden implikant (aby nie zmienić mocy pokrycia).

Przykład:

Na rys. 6.23a i b pogrubioną obwódką zaznaczyliśmy zwijane implikanty. W obu przypadkach zwinięty implikant powstał po obliczeniu części wspólnej zwijanego implikantu α i dopełnienia $F \setminus \{\alpha\}$. Zauważmy, że zwinięcie implikantu z rys. 6.23b daje w wyniku dwa implikanty, co powoduje zwiększenie mocy pokrycia. Oczywiście takie obrazkowe przedstawienie przydatne jest tylko do wytłumaczenia istoty zagadnienia.



Rys. 6.23 Ilustracja operacji zwijania

W programie Espresso algorytm zwijania sprowadza się do wykonania dwóch poniższych kroków:

Krok 1. Wyliczenia wag dla każdego implikantu (tak samo jak w procedurze EXPAND). Rozpoczęcia wykonywania kroku 2 dla implikantów z większymi wagami (czyli odwrotnie niż w procedurze EXPAND).

Krok 2. Zastąpienie każdego implikantu α maksymalnie zwiniętym implikantem $\alpha\alpha$ wyliczonym zgodnie z następującym wzorem:

$$\alpha\alpha = \text{superkostka}(\alpha \cap \overline{Q_\alpha}) = \alpha \cap \text{superkostka}(\overline{Q_\alpha})$$

gdzie: zbiór $Q = (\text{ON} \cup \text{DC}) \setminus \{\alpha\}$,

Q_α - kofaktor zbioru Q względem implikantu α

$\overline{Q_\alpha}$ - dopełnienie kofaktora Q_α

$\text{superkostka}(\overline{Q_\alpha})$ - najmniejsza kostka pokrywająca kostki zbioru $\overline{Q_\alpha}$

\cap - część wspólna implikantów

Przypomnijmy, że kofaktor implikantu α ze względu na implikant β jest pusty, jeśli α nie przecina β . W przeciwnym przypadku jest określany wzorem:

$$\alpha_\beta = \alpha_1 + \beta_1 \quad \alpha_2 + \beta_2 \quad \dots \quad \alpha_n + \beta_n$$

Kofaktor zbioru Q względem implikantu jest zbiorem odpowiednich kofaktorów.

Przykład:

Niech będzie dane pokrycie $F = \{\alpha = 11 \ 11 \ 10, \beta = 10 \ 11 \ 11, \gamma = 11 \ 01 \ 11\}$. Dla każdego implikantu z pokrycia będziemy wykonywać jego zwijanie. Na początek wyliczamy wagi każdego implikantu w identyczny sposób jak w procedurze EXPAND. W naszym przypadku waga każdego implikantu jest identyczna i wynosi 13, zatem możemy rozpocząć od dowolnej zmiennej (zgodnie z regułą powyżej zaczęlibyśmy od implikantu z największą wagą).

Zwijamy implikant α .

$$Q = \{\beta = 10 \ 11 \ 11, \gamma = 11 \ 01 \ 11\}$$

$$Q_\alpha = \left\{ \begin{matrix} 10 + 00 & 11 + 00 & 11 + 01 \\ 11 + 00 & 01 + 00 & 11 + 01 \end{matrix} \right\} = \left\{ \begin{matrix} 10 & 11 & 11 \\ 11 & 01 & 11 \end{matrix} \right\}$$

Wykorzystując procedurę COMPLEMENT otrzymujemy:

$$\overline{Q_\alpha} = \{01 \ 10 \ 11\}$$

Superkostka pojedynczego implikantu jest nim samym, dlatego

$$\alpha\alpha = \alpha \cap \text{superkostka}(\overline{Q_\alpha}) = \{11 \ 11 \ 10\} \cap \{01 \ 10 \ 11\} = \{01 \ 10 \ 10\}$$

Zwijamy implikant β .

$$Q = \{\alpha\alpha = 01 \ 10 \ 10, \gamma = 11 \ 01 \ 11\}$$

$$Q_\beta = \left\{ \begin{matrix} 01 + 01 & 10 + 00 & 10 + 00 \\ 11 + 01 & 01 + 00 & 11 + 00 \end{matrix} \right\} = \left\{ \begin{matrix} 01 & 10 & 10 \\ 11 & 01 & 11 \end{matrix} \right\} \quad \overline{Q_\beta} = \left\{ \begin{matrix} 10 & 10 & 11 \\ 11 & 10 & 01 \end{matrix} \right\}$$

$$\text{Zatem } \beta\beta = \beta \cap \text{superkostka}(\overline{Q_\beta}) = 10 \ 11 \ 11 \cap 11 \ 10 \ 11 = 10 \ 10 \ 11$$

Zwijamy implikant γ .

$$Q = \{\alpha\alpha = 01 \ 10 \ 10 \ \beta\beta = 10 \ 10 \ 11\}$$

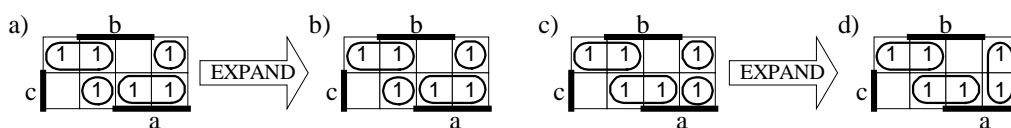
$$Q_\gamma = \left\{ \begin{matrix} 01 & 10 & 10 \\ 10 & 10 & 11 \end{matrix} \right\} \quad \overline{Q_\gamma} = U \text{ (czyli cała przestrzeń)}$$

$$\gamma\gamma = \gamma \cap \text{superkostka}(\overline{Q_\gamma}) = \gamma = 11 \ 01 \ 11$$

Zachęca się do zilustrowania powyższych obliczeń za pomocą tablic Karnaugh'a.

Przykład:

Kolejny przykład pokazuje jakie niespodzianki sprawia procedura REDUCE. Niech będzie dane pokrycie $F = \{\alpha = 11 \ 10 \ 10 \ \beta = 10 \ 11 \ 10 \ \gamma = 01 \ 11 \ 01 \ \delta = 11 \ 01 \ 01\}$. Wagi każdego implikantu są jednakowe i wynoszą 11. Zatem proces zwiwania możemy rozpocząć od dowolnego implikantu. Jeżeli zwiniemy najpierw δ , potem α i kolejne implikanty w dowolnej kolejności to otrzymamy pokrycie $F = \{\alpha\alpha = 01 \ 10 \ 10 \ \beta = 10 \ 11 \ 10 \ \gamma = 01 \ 11 \ 01 \ \delta\delta = 10 \ 01 \ 01\}$ przedstawione na rys. 6.24a. Gdy spróbujemy zwinąć najpierw α , potem γ i pozostałe implikanty to otrzymamy następujące pokrycie $F = \{\alpha\alpha = 01 \ 10 \ 10 \ \beta = 10 \ 11 \ 10 \ \gamma\gamma = 01 \ 10 \ 01 \ \delta = 11 \ 01 \ 01\}$ przedstawione na rys. 6.24c. Przyglądając się obu wynikom zwiwania na pierwszy rzut oka nie zauważamy żadnej niespodzianki. Po procedurze REDUCE wywoływana jest procedura EXPAND, która dla zwiwania z rys. 6.24a da w wyniku pokrycie z rys. 6.24b (nic się nie zmieniło) a dla przypadku z rys. 6.24c da pokrycie przedstawione na rys. 6.24d.



Rys. 6.24 a) zły wynik REDUCE b) dobry wynik REDUCE

Procedura IRREDUNDANT.

Procedura usuwania nadmiarowych implikantów z pokrycia jest wywoływana po procedurze EXPAND, ponieważ w wyniku rozwijania otrzymujemy listę implikantów prostych (być może nie wszystkich) lecz takich, że żaden z nich nie zawiera się w innym, jednak ich suma może pokrywać jakiś implikant. Zatem - jak nietrudno się domyśleć - celem procedury będzie usunięcie z pokrycia prostego F jak największej liczby implikantów nadmiarowych. Procedura dzieli początkowe pokrycie proste na trzy zbiory:

1. Zbiór implikantów względnie zasadniczych E , zawierający te implikanty, które pokrywają niektóre iloczyny zupełne funkcji, nie pokryte przez inne implikanty,
2. Zbiór całkowicie nadmiarowy T , zawierający te implikanty, które są pokrywane przez zbiór względnie zasadniczy, oraz
3. Zbiór częściowo nadmiarowy P , zawierający pozostałe implikanty.

Na początku wyznaczamy zbiór E . Wydaje się to łatwe, ponieważ wystarczy sprawdzić, czy implikant α nie jest pokryty przez zbiór $Q = \{F \cup DC\} \setminus \{\alpha\}$. Jeżeli tak jest, muszą istnieć jedyne funkcji nie pokryte przez zbiór Q , ale pokryte przez implikant α - czyli implikant α włączamy do zbioru E .

Wyznaczenie zbioru T polega to na sprawdzaniu, czy implikant należący do pierwotnego pokrycia jest pokryty przez $E \cup DC$. Jeżeli tak – implikant włączamy do zbioru T .

Ostatecznie, zgodnie z definicją, zbiór częściowo nadmiarowy $P = F \setminus \{E \cup T\}$.

Zatem głównym zadaniem procedury IRREDUNDANT będzie znalezienie podzbioru P , który razem z E pokrywa zbiór F .

Do wyznaczenia zbioru E implikantów względnie zasadniczych, skorzystamy z twierdzenia, które mówi, że zbiór implikantów Q pokrywa implikant α wtedy i tylko wtedy, gdy implikanty kofaktora Q_α wyznaczają wyrażenie boolowskie tożsamościowo równe jedynce logicznej, dla każdego zestawu wartości zmiennych wejściowych. Funkcję opisywaną powyższym wyrażeniem boolowskim nazywamy tautologią.

Inaczej mówiąc, implikant α jest pokrywany przez zbiór implikantów Q wtedy i tylko wtedy, gdy kofaktor Q względem α jest tautologią.

Poniżej przytoczymy reguły pozwalające określić wprost czy pewien zbiór kostek jest czy nie jest tautologią:

1. F jest tautologią, jeśli ma kostkę zawierającą same jedynki,
2. F jest tautologią, jeśli zależy tylko od jednej zmiennej i nie ma kolumny zer w tym polu,
3. F nie jest tautologią, jeśli ma kolumnę zawierającą same zera.

Jeżeli nie możemy stwierdzić wprost czy pokrycie jest albo nie jest tautologią, należy podzielić pokrycie ze względu na wybraną zmienną i dokonać sprawdzania tautologii dla obu pokryw z osobna (zgodnie z twierdzeniem, że $F=1 \Leftrightarrow (F_x=1) \wedge (F_{\bar{x}}=1)$).

Jeśli implikant α nie jest pokryty przez zbiór $Q = \{F \cup DC\} \setminus \{\alpha\}$, to należy go zaliczyć do zbioru E , czyli należy sprawdzać czy tautologią jest Q_α .

Przykład:

Niech będzie dane pokrycie $F=\{\alpha=10\ 10\ 11, \beta=10\ 11\ 01, \chi=11\ 01\ 01, \delta=01\ 01\ 11, \varepsilon=01\ 11\ 10\}$.

$$Q_{\alpha} = \left\{ \begin{array}{l} \beta\ 10\ 11\ 01 \\ \chi\ 11\ 01\ 01 \\ \delta\ 01\ 01\ 11 \\ \varepsilon\ 01\ 11\ 10 \end{array} \right\}_{10\ 10\ 11} = \{11\ 11\ 01\}$$

Nie jest tautologią.

$$Q_{\beta} = \left\{ \begin{array}{l} \alpha\ 10\ 10\ 11 \\ \chi\ 11\ 01\ 01 \\ \delta\ 01\ 01\ 11 \\ \varepsilon\ 01\ 11\ 10 \end{array} \right\}_{10\ 11\ 01} = \left\{ \begin{array}{l} \alpha\ 11\ 10\ 11 \\ \chi\ 11\ 01\ 11 \end{array} \right\}$$

Jest tautologią.

$$Q_{\chi} = \left\{ \begin{array}{l} \alpha\ 10\ 10\ 11 \\ \beta\ 10\ 11\ 01 \\ \delta\ 01\ 01\ 11 \\ \varepsilon\ 01\ 11\ 10 \end{array} \right\}_{11\ 01\ 01} = \left\{ \begin{array}{l} \beta \rightarrow 10\ 11\ 11 \\ \delta \rightarrow 01\ 11\ 11 \end{array} \right\}$$

Jest tautologią.

$$Q_{\delta} = \left\{ \begin{array}{l} \alpha\ 10\ 10\ 11 \\ \beta\ 10\ 11\ 01 \\ \chi\ 11\ 01\ 01 \\ \varepsilon\ 01\ 11\ 10 \end{array} \right\}_{01\ 01\ 11} = \left\{ \begin{array}{l} \chi \rightarrow 11\ 11\ 01 \\ \varepsilon \rightarrow 11\ 11\ 10 \end{array} \right\}$$

Jest tautologią.

$$Q_{\varepsilon} = \left\{ \begin{array}{l} \alpha\ 10\ 10\ 11 \\ \beta\ 10\ 11\ 01 \\ \chi\ 11\ 01\ 01 \\ \delta\ 01\ 01\ 11 \end{array} \right\}_{01\ 11\ 10} = \{\delta \rightarrow 11\ 01\ 11\}$$

Nie jest tautologią.

Zatem $E=\{\alpha, \varepsilon\}$. Żaden implikant nie jest pokrywany przez implikanty należące do E, dlatego $T=\emptyset$ oraz $P=\{\beta, \chi, \delta\}$. Aby najlepiej wybrać podzbiór zbioru P, który razem z E będzie minimalnym pokryciem, zbudujemy macierz pokrycia. Macierz ta ma tyle kolumn ile jest implikantów w zbiorze P. Każdy wiersz reprezentuje podzbiór kostek należących do P, których usunięcie spowoduje brak pokrycia fragmentu rozważanego implikantu. Czyli dla naszego przykładu macierz pokrycia jest następująca:

β	χ	δ

Pierwszy wiersz odpowiada Q_β , który jest liściem drzewa dekompozycji tautologii. Dwie kostki Q_β są związane z implikantami α i χ . Pierwszy jest kostką względnie zasadniczą (i tak już wejdzie do pokrycia), dlatego istotniejszą informacją jest to, że po usunięciu χ implikant β nie będzie pokryty (wpisaliśmy 1 w kolumnie χ). W kolumnie β wpisaliśmy też 1, bo β pokrywa sam siebie.

Drugi wiersz wynika z Q_χ . Wpisaliśmy jedynki w kolumnach β , δ , bo implikant χ pokrywa fragmenty implikantów β , δ (wpisaliśmy 1 także w kolumnie χ , która pokrywa sama siebie).

Analogicznie wypełniliśmy trzeci wiersz na podstawie Q_δ .

W tym momencie musimy rozwiązać klasyczny problem pokrycia, czyli pokryć wszystkie wiersze jedynkami z jak najmniejszej liczby kolumn. Widzimy, że kostka χ pokrywa wszystkie trzy wiersze, stąd w skład minimalnego pokrycia wchodzi $E \cup \{\chi\} = \{\alpha, \epsilon, \chi\}$.

Procedura ESSENTIALS

Jeżeli w zadanym pokryciu początkowym występują implikanty zasadnicze, to one muszą wejść do wynikowego pokrycia prostego i nienadmiarowego. Zatem nie ma sensu wykonywać na tych implikantach operacji rozwijania i zwijania, ponieważ i tak nie uzyskamy lepszego pokrycia. W metodzie Espresso na samym początku poszukuje się implikantów zasadniczych, wyłącza się je z pokrycia i przenosi do zbioru nieokreśloności. Dopiero po wyjściu z pętli głównej programu do uzyskanego pokrycia są dodawane wszystkie implikanty zasadnicze.

Poszukiwanie implikantów prostych oprzemy na następującym wniosku:

Niech ON będzie pokryciem zbioru włączenia, DC pokryciem zbioru nieokreśloności a α implikantem prostym. Wówczas α jest implikantem zasadniczym wtedy i tylko wtedy, gdy $H \cup ON$ nie pokrywa α , gdzie

$$H = \text{consensus}(((ON \cup DC) \# \alpha), \alpha)$$

gdzie operator $\#$ oznacza operację wyostrażania a operator $\text{consensus}(\alpha, \beta)$ między implikantami α oraz β jest obliczany według wzoru:

$$\text{consensus}(\alpha, \beta) = \begin{cases} \alpha_1 + \beta_1 & \alpha_2 \bullet \beta_2 \dots \alpha_n \bullet \beta_n \\ \alpha_1 \bullet \beta_1 & \alpha_2 + \beta_2 \dots \alpha_n \bullet \beta_n \\ \alpha_1 \bullet \beta_1 & \alpha_2 \bullet \beta_2 \dots \alpha_n + \beta_n \end{cases}$$

6.5 Program ESPRESSO

Espresso jest programem konsolowym, wywoływanym z linii poleceń w następujący sposób:

Espresso [*opcje*] [*plik*]

gdzie *plik* jest plikiem wejściowym programu, którego zawartość określa funkcje przełączające podana jest w odpowiednim standardzie, zaś *opcje* dotyczą sposobu przetwarzania.

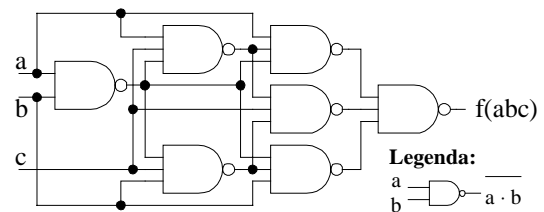
Program z pełnym opisem opcji oraz formatem pliku zawartości wejściowego można znaleźć w internecie.

Literatura

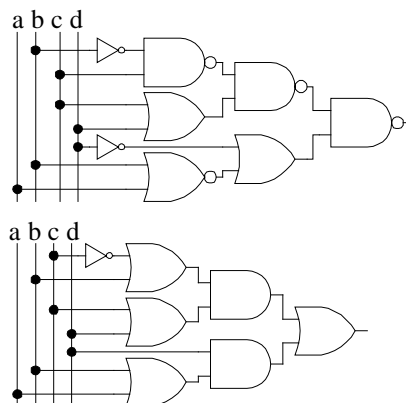
- [1] Karnaugh M., A Map method for Synthesis of Combinational Logic Circuits, Trans. AIEE, Communications and Electronics, 72, Nov. 1953, str.593-599,
- [2] Veitch E.W., A Chart Method for Simplifying Boolean Functions, Proc. of ACM, May 1952, str. 127-133
- [3] McCluskey E.J., Minimization of Boolean Functions, Bell Sys. Tech. J., Vol. 35, No. 5, Nov. 1956, str. 1417-1444
- [4] Quine W.V., The problem of simplifying truth functions, American Mathematical Monthly, Vol.59, Oct. 1952, str. 521-531
- [5] Quine W.V., A way to simplify truth functions, American Mathematical Monthly, Vol. 62, Nov. 1955, str. 627-631.
- [6] Wakerly J.F., Digital design principles and practices, 2nd edition, Prentice-Hall, 1994
- [7] Yang S., Logic synthesis and optimization benchmarks user guide, Microelectronics Center of North Carolina, January 1991.
- [8] McGeer P., Sanghavi J., Brayton R., Sangiovanni-Vincentelli A., ESPRESSO-SIGNATURE: A new exact minimizer for logic functions, Proc. of the Design Automation Conference, 1993, str. 618-621.
- [9] Coudert O., Madre J., Fraisse H., A new viewpoint on two-level logic minimization, Proc. of the Design Automation Conference, 1993, str. 625-630

ĆWICZENIA

1. Podać realizację układu kombinacyjnego działającego w następujący sposób - jeżeli na wejście X podamy binarny odpowiednik liczby dziesiętnej K, $0 \leq K \leq 7$, to na wyjściu Z otrzymamy binarne przedstawienie minimalnej liczby całkowitej $\geq \sqrt{K}$.
2. Podać minimalną postać dysjunkcyjną funkcji realizowanej przez układ z rys.6.17, wykorzystując w tym celu przekształcenia formalne i tablice Karnaugh.
3. Wykaż, że układy przedstawione na rys. 6.18a i b realizują tę samą funkcję przełączającą. Czy można pokazać równoważność układów nie wykorzystując postaci kanonicznych, tablicy Karnaugh bądź wyrażeń boolowskich?
4. Podaj minimalną postać dysjunkcyjną funkcji przełączających, wykorzystując w tym celu tablice Karnaugh:
 - a) $f(abcd) = \Sigma(1,4,5,6,7,10,11,13)$,
 - b) $f(abcd) = \Sigma(0,2,3,8,9,12,14,15)$,
 - c) $f(abcd) = \Sigma(1,4,5,11,14,15)$,



Rys. 6.25 Przykładowy układ kombinacyjny



Rys. 6.26 Przykład równoważnych układów kombinacyjnych

- d) $f(abcd) = \Sigma(1,2,4,7,8,11,13,14)$,
- e) $f(abcd) = \Sigma(1,3,4,5,6,7,8,9,10,12,13,14,15)$,
- f) $f(abcd) = \Sigma(0,2,4,5,6,7,8,9,10,11,13)$.

5. Podaj minimalną postać dysjunkcyjną następujących nie w pełni określonych funkcji przełączających, wykorzystując w tym celu tablice Karnaugh:

- a) $f(abcd) = \Sigma(1,5) + \Sigma_{\emptyset}(7,9,11,15)$,
- b) $f(abcd) = \Sigma(7,11,13,14,15) + \Sigma_{\emptyset}(0,2,8,10)$,
- c) $f(abcd) = \Sigma(3,5,8,9,15) + \Sigma_{\emptyset}(1,7,13)$,
- d) $f(abcd) = \Sigma(2,6,9,15) + \Sigma_{\emptyset}(3,11)$.

Cyfra	Kod BCD abcd	Cyfra	Kod BCD abcd
0	0000	5	0101
1	0001	6	0110
2	0010	7	0111
3	0011	8	1000
4	0100	9	1001

6. Pewien układ kombinacyjny ma cztery wejścia a, b, c oraz d, na które podajemy pobudzenia odpowiadające binarnie kodowanym cyframi dziesiętnym (kod BCD, *binary coded decimal*) zgodnie z rys. 6.19. Możliwych jest tylko 10 pobudzeń, ponieważ mamy tylko 10 cyfr dziesiętnych. Pozostałych 6 pobudzeń nigdy nie zostanie podanych na wejścia układu. Przy tak podanych założeniach, podaj minimalną realizację układu, który na swoim wyjściu sygnalizuje pojawienie się na wejściu cyfr dziesiętnych 7,8,9 w kodzie BCD.
7. Zsyntezuj kombinacyjny układ o trzech wyjściach f_1 , f_2 oraz f_3 , na którego wejścia podajemy liczby w kodzie BCD. $f_1=1$ tylko wtedy, gdy podana liczba jest podzielna bez reszty przez 3; $f_2=1$, gdy liczba jest większa od 4; $f_3=1$, gdy liczba należy do przedziału [4-8].
8. Dwubitowym układem porównującym nazwiemy układ, na który podajemy dwie dwubitowe liczby $A=a_2a_1$ oraz $B=b_2b_1$, a na jego wyjściu otrzymujemy wartość 1 wtedy i tylko wtedy, gdy $A>B$. Podaj minimalną realizację układu.
9. Powtórz zadanie 8 dla przypadku trzybitowego układu porównującego.
10. Zaprojektuj układ o trzech wyjściach f_1 , f_2 , f_3 , dwóch wejściach informacyjnych x_1, x_2 oraz dwóch wejściach sterujących S_1, S_2 . Wartości zmiennych sterujących decydują o postaci funkcji wyjściowych zgodnie z rys. 6.20.
11. Spróbuj podać algorytm generowania funkcji, która posiada dokładnie 2, 3, 4,... minimalne postacie dysjunkcyjne.
12. Pokaż, że wykonanie kroków algorytmu iteracyjnego konsensusu prowadzi do wygenerowania implikantów prostych.
13. Pokaż, że istnieje konsensus dla dwóch implikantów $a_1a_2...a_n$ oraz $c_1c_2...c_n$ (w notacji pozycyjnej), wtedy i tylko wtedy, gdy dla pewnego k , $1 \leq k \leq n$, $a_k = \bar{c}_k$ oraz $a_i \neq \bar{c}_i$, $i \neq k$.
14. Przypomnijmy, że operacja wyostrzania to $A \# B = A - (A \cap B)$. Niech $IP = \{P1, P2, P3, P4\}$ będzie zbiorem wszystkich implikantów prostych pewnej funkcji. Obliczmy
- $$S = P1 \# \{P2, P3, P4\} = (((P1 \# P2) \# P3) \# P4)$$
- Pokaż, że jeżeli zbiór S nie jest pusty to $P1$ jest zasadniczym implikantem prostym.
15. Niech będzie dana funkcja $f(abcdekmo) = oc + okm + ode + abkm + abc + abde$, którą można zrealizować w dwupoziomowym układzie za pomocą sześciowejściowej bramki OR, trzech trójwejściowych bramek AND, dwóch czterowejściowych bramek AND oraz jednej dwuwejściowej bramki AND. Pokaż, że można zrealizować powyższą funkcję za pomocą czterech dwuwejściowych bramek AND, jednej dwuwejściowej oraz jednej trójwejściowej bramki OR.
16. Czy poniższy algorytm pozwala otrzymać minimalną postać funkcji funkcji?
- a) zaznaczyć w tablicy Karnaugh'a wszystkie implikanty proste,
 - b) Wybrać do postaci minimalnej wszystkie zasadnicze implikanty proste,
 - c) Wybrać minimalny podzbiór z pozostałych implikantów, które pokrywają pozostałe jedynki nie pokryte przez implikanty zasadnicze.

S_1S_2	f_1	f_2	f_3
00	$x_1 \oplus x_2$	x_1	x_2
01	1	0	x_1
10	1	x_2	0
11	$x_1 \oplus x_2$	0	0

Rys. 6.28 Specyfikacja funkcji dla układu z zadania 10

7 SYSTEM FUNKCJONALNIE PEŁNY

W rozdziale wprowadzającym do kombinacyjnych układów cyfrowych powiedzieliśmy, że dowolną funkcję przełączającą możemy zrealizować jedynie za pomocą bramek AND, OR oraz NOT, ponieważ do uzyskania realizacji dowolnego wyrażenia boolowskiego wystarczy, by realizowane były operacje mnożenia, dodawania i dopełnienia.

Zapytajmy więc:

1. Czy można zrealizować dowolną funkcję przełączającą używając innego zestawu bramek? Jeśli odpowiedź byłaby twierdząca, to oczywiste jest drugie pytanie,
2. Jakie własności powinny mieć funkcje realizowane przez bramki należące do jednego zestawu?

Zestaw bramek (w szczególności może to być jedna bramka) za pomocą którego można zrealizować wszystkie możliwe funkcje przełączające nazywamy systemem funkcjonalnie pełnym (w skrócie SFP). W roku 1921 E. Post [1] odpowiedział twierdząco na pytanie pierwsze, przy okazji formułując twierdzenie dające odpowiedź na pytanie drugie.

7.1 Twierdzenie Posta

Własności funkcji, które wykorzystamy formułując twierdzenie Posta, są następujące:

1. Funkcja zachowuje stałą zero, gdy $f(0,0,...,0)=0$,
2. Funkcja zachowuje stałą jeden, gdy $f(1,1,...,1)=1$,
3. Funkcja jest samodwoista, gdy $f(x_1, x_2, ..., x_n) = \overline{f(\overline{x}_1, \overline{x}_2, ..., \overline{x}_n)}$.

Funkcję $\overline{f(\overline{X})}$ nazywamy dualną i oznaczamy $f^d(X)$.

4. Funkcja jest monotoniczna, gdy $(\forall X, Y)(X \leq Y) \Rightarrow (f(X) \leq f(Y))$, gdzie $X=(x_1, x_2, ..., x_n)$, $Y=(y_1, y_2, ..., y_n)$, $x_i, y_i \in \{0,1\}$ a relacja $X \leq Y$ oznacza, że $x_i \leq y_i$ dla $\forall i$. Oczywiście przyjmujemy, że $0 \leq 0$, $0 \leq 1$, $1 \leq 1$. Gdy dla pewnego i , $x_i < y_i$ to będziemy oznaczać ten fakt przez $X < Y$.

5. Funkcja jest liniowa, gdy możemy przedstawić ją w postaci:

$$f(x_1, x_2, ..., x_n) = c_0 \oplus c_1 x_1 \oplus c_2 x_2 \oplus ... \oplus c_n x_n,$$

gdzie $c_i \in \{0,1\}$, $0 \leq i \leq n$, $x \oplus y = x \cdot \overline{y} + \overline{x} \cdot y$.

Twierdzenie 7.1.1 (Posta)

Zbiór funkcji przełączających $\{f_1, f_2, ..., f_n\}$ tworzy system funkcjonalnie pełny wtedy i tylko wtedy, gdy zbiór ten zawiera funkcje:

- a) nie zachowujące stałej zero,
- b) nie zachowujące stałej jeden,
- c) nie będące samodwoistą,
- d) nie będące monotoniczną
- e) nie będące liniową.

Przykładowo, jeżeli w rozważanym zbiorze funkcji nie ma funkcji, która nie zachowuje stałej zero, to zbiór nie jest SFP. Jeżeli pewna funkcja spełnia wszystkie własności a-e, ona jedna tworzy SFP (cenna własność!).

W przykładzie przedstawiono funkcje przełączające, które spełniają (bądź nie) podane wyżej własności, a w tab. 7.1 podano własności dla wszystkich szesnastu funkcji dwóch zmiennych.

Przykład:

Niech będzie dana funkcja $f(x_1 x_2 x_3) = \overline{x}_1 \overline{x}_2 x_3 + \overline{x}_1 x_2 \overline{x}_3 + x_1 x_2 x_3 + x_1 \overline{x}_2 \overline{x}_3$, która jest:

liniowa: $f(x_1 x_2 x_3) = x_1 \oplus x_2 \oplus x_3,$

samodwoista: $\overline{f(x_1 x_2 x_3)} = \overline{x_1} \overline{x_2} \overline{x_3} + \overline{x_1} x_2 x_3 + x_1 x_2 \overline{x_3} + x_1 \overline{x_2} x_3$

$$\overline{f(\overline{x_1} \overline{x_2} \overline{x_3})} = x_1 x_2 x_3 + x_1 \overline{x_2} \overline{x_3} + \overline{x_1} x_2 x_3 + \overline{x_1} \overline{x_2} \overline{x_3} = f(x_1 x_2 x_3),$$

zachowuje zero: $f(0,0,0)=0,$

zachowuje jeden: $f(1,1,1)=1,$

Funkcja nie jest monotoniczna, ponieważ dla $X=(100)$ oraz $Y=(101)$ (czyli $X \leq Y$) nie jest prawdą, że $f(100) \leq f(101)$.

Przykład:

Rozważmy funkcję $f(x_1 x_2 x_3) = \overline{x_1 x_2 x_3} = \overline{x_1} + \overline{x_2} + \overline{x_3}.$


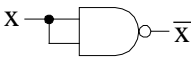
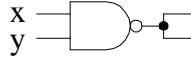
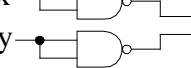

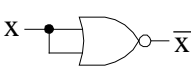
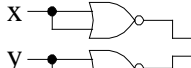
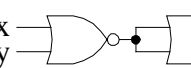
Funkcja nie jest samodwoista, nie jest monotoniczna, nie jest liniowa oraz nie zachowuje zera i jedności.

Tabela 7.1

Tabela 7.1

Funkcje przełączające dwóch zmiennych																
Własność	a	\bar{a}	b	\bar{b}	0	1	ab	a+b	$\bar{a}\bar{b}$	$a\bar{b}$	$a \oplus b$	$a \uparrow b$	$a \equiv b$	$\bar{a} + b$	$a + \bar{b}$	$a b$
Zachowuje zero	T	N	T	N	T	N	T	T	T	T	T	N	N	N	N	N
Zachowuje jeden	T	N	T	N	N	T	T	T	N	N	N	N	T	T	T	N
Samodwoista	T	T	T	T	N	N	N	N	N	N	N	N	N	N	N	N
Monotoniczna	T	N	T	N	T	T	T	T	N	N	N	N	N	N	N	N
Liniowa	T	T	T	T	T	T	N	N	N	N	T	N	T	N	N	N
nazwa elementu		NOT		NOT			AND	OR			EOR	NOR	ENOR			NAND
numer funkcji	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
T - tak, N - nie																

Twierdzenie Posta jest twierdzeniem ogólnym pozwalającym odpowiedzieć na pytanie: czy dany zbiór funkcji tworzy zbiór funkcjonalnie pełny?. Istnieje prostszy sposób udowodnienia pełnej funkcjonalności. Otóż jeżeli wiemy, że $\{\text{AND}, \text{OR}, \text{NOT}\}$ tworzy zbiór funkcjonalnie pełny, to nic nie stoi na przeszkodzie, abyśmy wykazali jedynie, że za pomocą funkcji rozważanego zbioru można zrealizować iloczyn, sumę i dopełnienie.

Element	NOT	AND	OR
NAND 			
NOR 			

Rys. 7.1 NAND i NOR jako systemy funkcjonalnie pełne

Z punktu widzenia producenta elementów logicznych korzystniejsze jest produkowanie jak najmniejszej liczby bramek. Wynika z tego potrzeba poszukiwania minimalnego zbioru funkcjonalnie pełnego, przy czym kryterium minimalizacyjne może być różnie rozumiane. Możemy poszukiwać minimalnego zbioru w ogóle, albo - jak to często bywa - znamy wybraną bramkę i pytamy, których bramek brakuje, by razem z wybraną zbiór był funkcjonalnie pełny i zawierał ich minimalną liczbę.

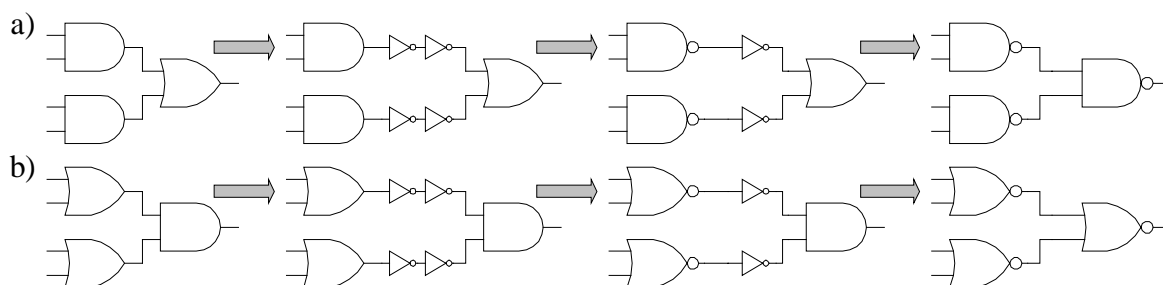
W tabeli 7.1 znajdujemy dwie ciekawe funkcje: strzałka Pierce'a, to funkcja numer 12 oraz kreska Sheffera, oznaczona numerem 16. Są to jedyne dwie funkcje w tabeli, dla których niespełnione są wszystkie własności. Dlatego każda z tych funkcji tworzy system funkcjonalnie pełny. Strzałka Pierce'a realizowana jest za pomocą bramki nazwanej NOR (NOT-

OR), zaś kreska Sheffera - bramki NAND (NOT-AND). Na rysunku 7.1 przedstawiliśmy sposób realizacji AND, OR i NOT z wykorzystaniem tylko powyższych elementów logicznych.

Grupa	System funkcjonalnie pełny
Tylko jeden element	{NAND} numer 16, {NOR} numer 12
Tylko jeden element oraz stała 0 lub 1	{zakaz przez x} numer 9 lub 10 {implikacja} numer 14 lub 15
Tylko dwa elementy	{AND, NOT}, {OR, NOT}, {EOR, AND, 1}

Rys. 7.2 Przykładowe systemy funkcjonalnie pełne

Na rysunku 7.2 pokazaliśmy trzy grupy systemów funkcjonalnie pełnych, szczególnie przydatnych w praktyce inżynierskiej.



Rys. 7.3 Rysunkowa ilustracja twierdzenia 7.2.1: a - przekształcenie układu AND-OR w układ NAND, b - przekształcenie układu OR-AND w układ NOR

Bardzo popularna jest bramka Exclusive-OR (EOR), której własności są szczególnie interesujące w układach syntezyowanych za pomocą technik spektralnych oraz w połączeniu z elementami wieloprogowymi. Z punktu widzenia globalnej minimalizacji (gdy zależy nam na mniejszej liczbie połączeń międzyelementowych), bramki EOR i elementy progowe powinny odgrywać ważną rolę w układach cyfrowych. O elementach progowych oraz własnościach operatora EOR będzie mowa w dalszych rozdziałach.

7.2 Układy NAND i NOR

Pod pojęciem układu NAND (NOR) będziemy rozumieć układ kombinacyjny złożony jedynie z bramek NAND (NOR), a dwupoziomową realizację układową dowolnej postaci dysjunkcyjnej (koniunkcyjnej) będziemy nazywać AND-OR (OR-AND).

Twierdzenie 7.2.1

Dowolna funkcja przełączająca zrealizowana w postaci AND-OR (OR-AND) pozostaje nie zmieniona, jeżeli każdą bramkę AND oraz OR zastąpimy bramkami NAND (NOR).

Dowód:

Prawdziwość twierdzenia udowodnimy na przykładzie najprostszej postaci dysjunkcyjnej i koniunkcyjnej:

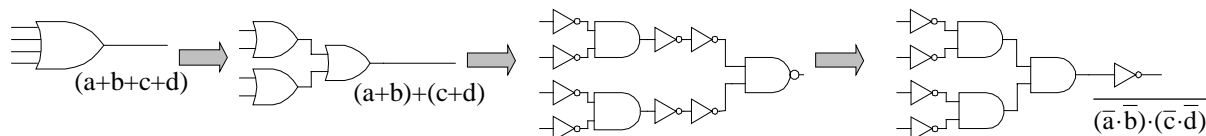
$$\begin{aligned}
 xy + wz &= \overline{\overline{xy + wz}} = \overline{\overline{xy} \cdot \overline{wz}} = \overline{(\overline{x} \mid \overline{y}) \cdot (\overline{w} \mid \overline{z})} = \overline{(\overline{x} \mid \overline{y})} \mid \overline{(\overline{w} \mid \overline{z})} \\
 (x + y)(w + z) &= \overline{\overline{(x + y)(w + z)}} = \overline{\overline{(x + y)} + \overline{\overline{(w + z)}}} = \overline{(\overline{x} \uparrow \overline{y}) + (\overline{w} \uparrow \overline{z})} = \overline{(\overline{x} \uparrow \overline{y})} \uparrow \overline{(\overline{w} \uparrow \overline{z})} \quad \text{c.n.p.}
 \end{aligned}$$

Twierdzenie 7.2.2

Jeżeli w realizacji OR-AND (AND-OR) każdy element OR oraz AND zastąpimy przez element NAND (NOR), otrzymamy realizację funkcji dualnej.

Dowód tego twierdzenia przebiega analogicznie jak twierdzenia 7.2.1.

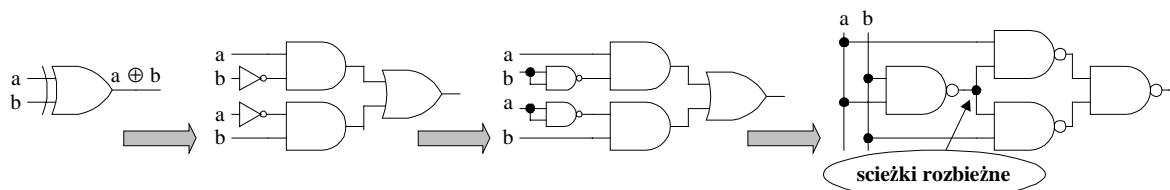
Powyższe twierdzenia zilustrujemy „metodą obrazkową” (rys.7.3.a,b). W praktyce dostępne są bramki z różną liczbą wejść. Na rysunku 7.4 pokazaliśmy sposób zastąpienia brakującej czterowejściowej bramki OR przez bramki dwuwejściowe.



Rys. 7.4 Równoważność układów

Zwróćmy uwagę, że zastąpienie bramek AND i OR w minimalnej realizacji AND-OR przez bramki NAND nie musi prowadzić do minimalnej realizacji NAND. Za przykład może służyć realizacja funkcji różnicy symetrycznej (EOR) za pomocą bramek NAND, uzyskanej wprost z realizacji AND-OR i realizacji minimalnej (rys. 7.5). Realizacja, którą nazwaliśmy minimalną jest najlepsza ze względu na następujące trzy warunki: $a \oplus b$

- Liczba elementów logicznych jest najmniejsza,
- Całkowita liczba wejść jest najmniejsza (przez wejście rozumiemy także dopełnienie zmiennej).
- Układ odpowiada ograniczeniom na istnienie ścieżek rozbieżnych (patrz rys. 7.5).



Rys. 7.5 Minimalna realizacja pEOR za pomocą układu NAND

Do chwili obecnej nie opracowano konstruktywnego algorytmu umożliwiającego minimalizację układów NAND realizujących dowolną funkcję zależną od n zmiennych.

7.3 Układy EOR - postać Reeda-Mullera

Obecnie przytoczymy kilka pożytecznych związków, które ułatwiają manipulację wyrażeniami zawierającymi operator Exclusive-OR (EOR). Zależności te powinny służyć przede wszystkim do minimalizacji układu realizującego zadaną funkcję przełączającą. Wykazanie prawdziwości podanych zależności pozostawia się Czytelnikowi (Tabela 7.2).

Tabela 7.2

1. $a \oplus b = b \oplus a$	10. $a + (a \oplus b) = a + b$
2. $a \oplus (b \oplus c) = (a \oplus b) \oplus c = a \oplus b \oplus c$	11. $(a \oplus b) \cdot (a \oplus c) = (a \oplus b) \cdot (b \oplus c) = (a \oplus c) \cdot (b \oplus c)$
3. $a \cdot (b \oplus c) = (a \cdot b) \oplus (a \cdot c)$	12. $(a \oplus b) + (a \oplus c) = (a \oplus b) + (b \oplus c) = (a \oplus c) + (b \oplus c)$
4. $a \oplus a = \bar{a} \oplus \bar{a} = 1 \oplus 1 = 0 \oplus 0 = 0$	13. $a \oplus (a \cdot b) = a \cdot \bar{b}$
5. $a \oplus \bar{a} = 1 \oplus 0 = 0 \oplus 1 = 1$	14. $a \oplus (\bar{a} \cdot b) = a + \bar{a}b = a + b$
6. $\overline{a \oplus b} = a \oplus b = \bar{a} \oplus \bar{b} = a \oplus \bar{b}$	15. $a \oplus (a + b) = \bar{a} \cdot b$
7. $a \oplus 0 = a \oplus 1 = a$	16. $a \oplus (\bar{a} + b) = \bar{a} + \bar{b}$
8. $a \oplus 1 = a \oplus 0 = \bar{a}$	17. $a \cdot f_1(X) \oplus \bar{a} \cdot f_2(X) = a \cdot f_1(X) + \bar{a} \cdot f_2(X)$, gdzie f_1, f_2 są dowolnymi funkcjami przełączającymi
9. $a \cdot (a \oplus b) = a \cdot \bar{b}$	

wierzchołków, dla których $f=0$. Ponadto, wszystkie wierzchołki $f=1$ leżą po jednej stronie płaszczyzny, a te, dla których $f=0$, leżą po stronie przeciwnej.

Przykład:

Trójwejściowy element większościowy wykorzystamy do zbudowania sumatora liczb dziesiętnych. Na wejścia sumatora podajemy dwa zestawy liczb dziesiętnych zakodowanych binarnie: $a_{n-1}a_{n-2}...a_1a_0$ oraz $b_{n-1}b_{n-2}...b_1b_0$.

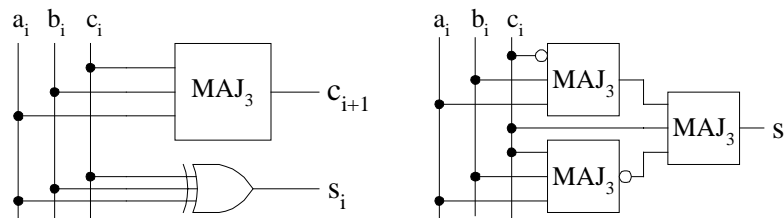
W wyniku dodawania arytmetycznego otrzymujemy sumę: $S_{n-1}S_{n-2}...S_1S_0$. Bity a_0, b_0, S_0 nazywamy młodszy, a $a_{n-1}, b_{n-1}, S_{n-1}$ - starszymi. Rozpoczynając dodawanie od najmłodszych bitów, otrzymujemy sumy częściowe S_i oraz przeniesienie na następną pozycję c_{i+1} :

$$S_i = a_i \oplus b_i \oplus c_i,$$

$$c_{i+1} = \text{MAJ}_3(a_i, b_i, c_i), \quad 0 \leq i \leq n,$$

gdzie c_i jest przeniesieniem wnoszonym na i -tą pozycję ($c_0=0$).

Przeniesienie do następnej pozycji $c_{i+1}=1$, gdy $a_i + b_i + c_i \geq 2$ oraz $c_{i+1}=0$, gdy $a_i + b_i + c_i < 2$ (Uwaga: $+$ oznacza dodawanie arytmetyczne). Przykładową realizację pojedynczego ogniwa sumatora przedstawiliśmy na rys.7.8.



Rys. 7.8 Przykładowe realizacje ogniwa sumatora wykorzystujące elementy większościowe

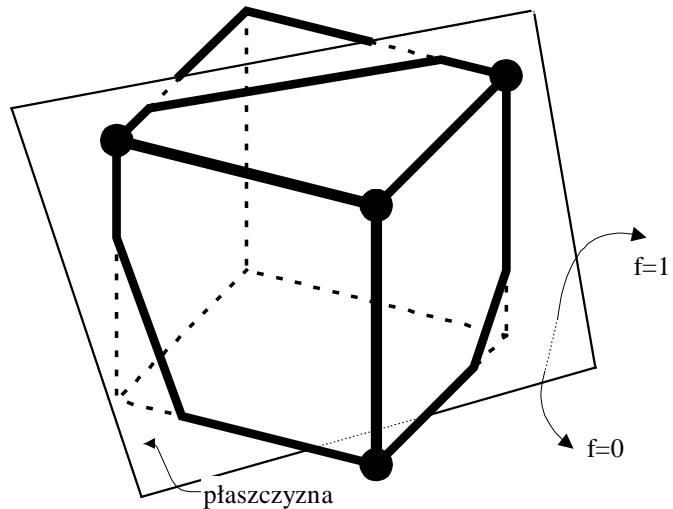
Bramka większościowa nie tworzy systemu funkcjonalnie pełnego, ponieważ nie możemy uzyskać za jej pomocą dopełnienia. Bramki AND i OR uzyskujemy tak, jak to pokazaliśmy na rys.7.9 (a po dopełnieniu bramki NAND i NOR).

Wykorzystanie bramek większościowych jest utrudnione, ponieważ dotychczas nie istnieje algebraiczna postać wykorzystująca operatory większościowe umożliwiające przedstawienie dowolnej funkcji przełączającej (np. taką postacią dla operatora EOR jest postać Reeda-Mullera).

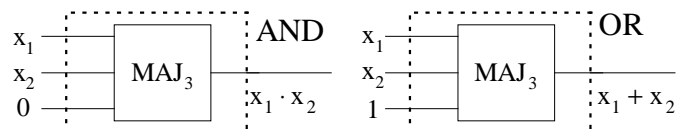
Zaproponowano jedynie postać, która wykorzystuje trójwejściowy operator większościowy. Dowolną funkcję $f(X)$ możemy za jego pomocą przedstawić następująco:

$$f(x_1, x_2, \dots, x_n) = 1 \# x_1 f(1, x_2, \dots, x_n) \# \bar{x}_1 f(0, x_2, \dots, x_n)$$

gdzie $\#$ reprezentuje trójwejściowy operator większościowy.



Rys. 7.7 Funkcja większościowa trzech zmiennych



Rys. 7.9 Realizacje podstawowych bramek wykorzystujące elementy większościowe

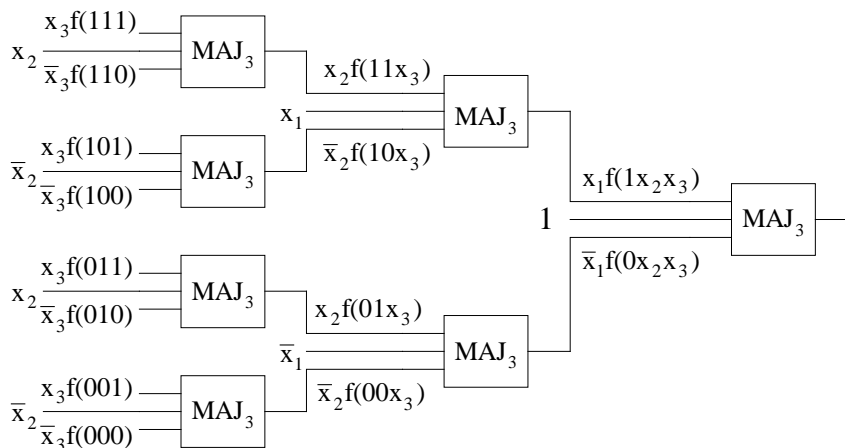
Każda z funkcji $x_1 f(1, x_2, \dots, x_n)$ i $\bar{x}_1 f(0, x_2, \dots, x_n)$ może być następnie przedstawiona jako:

$$x_1 f(1, x_2, \dots, x_n) = x_1 \# x_2 f(1, 1, \dots, x_n) \# \bar{x}_2 f(1, 0, \dots, x_n)$$

oraz

$$\bar{x}_1 f(0, x_2, \dots, x_n) = \bar{x}_1 \# x_2 f(0, 1, \dots, x_n) \# \bar{x}_2 f(0, 0, \dots, x_n)$$

Dokonujemy powyższego rozwinięcia do chwili, aż otrzymamy funkcje większościowe z wejściami x_n i \bar{x}_n . Na rysunku 7.10 przedstawiliśmy układ realizujący dowolną funkcję przełączającą trzech zmiennych.



Rys. 7.10 Układ realizujący dowolną funkcję trzech zmiennych

7.5 Funkcje progowe

Definicja 7.5.1

Funkcją progową nazywamy funkcję przełączającą realizowaną przez pojedynczy element progowy taki, że

$$f(x_1, x_2, \dots, x_n) = 1 \Leftrightarrow \sum_{i=1}^n w_i \cdot x_i \geq T$$

$$f(x_1, x_2, \dots, x_n) = 0 \Leftrightarrow \sum_{i=1}^n w_i \cdot x_i \leq T-1$$

gdzie: x_i - zmienne wejściowe,
 w_i - całkowitoliczbowe wartości nazywane wagami,
 T - wartość nazywana progiem,
 $-, \sum, \cdot$ - odpowiednio, **arytmetyczne** odejmowanie, sumowanie i mnożenie.

Dowolne realizacje funkcji przełączających wymagają mniejszej liczby elementów progowych oraz połączeń międzyelementowych niż realizacje wykorzystujące konwencjonalne elementy AND, NAND, itp.

Przykład:

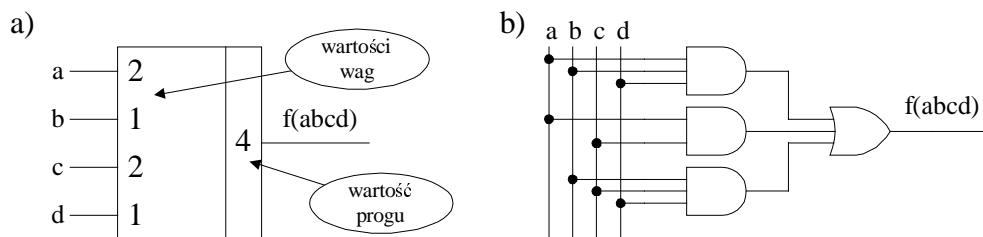
Niech będzie dana funkcja $f(abcd) = ac + abd + bcd$. Funkcja ta jest funkcją progową, ponieważ realizuje ją pojedynczy element progowy (rys.7.11a). Na rysunku 7.11b pokazaliśmy realizację danej funkcji przy użyciu konwencjonalnych bramek. Na uwagę zasługuje fakt, że połączeń międzyelementowych w realizacji konwencjonalnej mamy 12, a z wykorzystaniem elementu progowego tylko 5.

W dalszej części podręcznika będziemy wykorzystywać zapis $f(X): [W;T]$, co oznaczać będzie, że funkcja $f(X)$ jest realizowana przez element progowy o strukturze $[W;T]$, gdzie W - wektor wartości wag, T - wartość progu.

Funkcje progowe nazywano także funkcjami linioworozdzielnymi. Wynikało to z faktu, że wyrażenie

$$\sum_{i=1}^n w_i \cdot x_i = T$$

określa hiperpłaszczyznę w przestrzeni n -wymiarowej, oddzielającą wierzchołki, w których funkcja przyjmuje wartość 1, od tych wierzchołków, dla których funkcja ma wartość 0. Jeżeli w przestrzeni n -wymiarowej potrafimy tak „ustawić” hiperpłaszczyznę, by wszystkie wierzchołki, dla których wartość funkcji równa się 1, znalazły się po jednej stronie hiperpłaszczyzny, a wartości, dla których $f(X)=0$ po stronie drugiej, to rozpatrywana funkcja $f(X)$ jest funkcją progową (zobacz też rys. 7.7).



Rys. 7.11 Realizacja przykładowej funkcji: a - z wykorzystaniem elementu progowego, b - z wykorzystaniem bramek podstawowych

Definicja 7.5.2

Jeżeli istnieje dysjunkcyjna postać funkcji, w której pewna zmienna występuje tylko bez dopełnienia (tylko z dopełnieniem), to mówimy, że funkcja jest jednorodna ze względu na tę zmienną. Funkcja przełączająca jednorodna ze względu na wszystkie zmienne nazywa się krótko jednorodną (*unate*).

Przykład:

Funkcja $f(abc) = a + b\bar{c}$ jest funkcją jednorodną, ponieważ w podanej postaci dysjunkcyjnej każda ze zmiennych pojawia się jako afirmacja albo jako dopełnienie.

Twierdzenie 7.5.1

Funkcja jednorodna ma dokładnie jedną minimalną postać dysjunkcyjną, zawierającą wszystkie implikanty proste.

Twierdzenie 7.5.2

Każda funkcja progowa jest jednorodna (lecz nie odwrotnie).

Dowód:

Z twierdzenia Shannona wiemy, że każdą funkcję przełączającą możemy rozwinąć do postaci:

$$f(x_1, x_2, \dots, x_n) = x_i f(x_1, \dots, x_{i-1}, 1, \dots, x_n) + \bar{x}_i f(x_1, \dots, x_{i-1}, 0, \dots, x_n) = x_i f_1 + \bar{x}_i f_0$$

Nietrudno pokazać, że jeżeli $f(X): [w_1, w_2, \dots, w_n : T]$, to

$$f_1 = f(x_1, \dots, x_{i-1}, 1, \dots, x_n) : [w_1, \dots, w_{i-1}, 0, \dots, w_n : T_1 = T - w_i]$$

oraz

$$f_0 = f(x_1, \dots, x_{i-1}, 0, \dots, x_n) : [w_1, \dots, w_{i-1}, 0, \dots, w_n : T].$$

W zależności od wartości wagi w_i (dodatnia, zero, ujemna) mogą być spełnione następujące warunki: $T_1 < T$, $T_1 = T$, $T_1 > T$.

1. Rozważmy przypadek $T_1 < T$. Wtedy $f_0 \subseteq f_1$, ponieważ struktura wag dla funkcji f_0 i f_1 jest identyczna i jeśli wzbudzenie $W \cdot X$ jest nie mniejsze od T , to tym bardziej jest ono nie mniejsze od T_1 . W tym przypadku nie istnieje zatem taki zestaw wartości zmiennych wejściowych X , dla których $f_1=0$ oraz $f_0=1$.
2. $T_1 > T$. Rozumując podobnie jak w przypadku 1, dojdziemy do wniosku, że $f_1 \subseteq f_0$.
3. $T_1 = T$. W tym przypadku $f_1 = f_0$.

Możliwe są zatem trzy następujące zależności między funkcjami:

1. $f_0 \subseteq f_1$
2. $f_1 \subseteq f_0$
3. $f_1 = f_0$.

Jeżeli $f_1 = f_0 = F$, to $f(X) = x_i f_1 + \bar{x}_i f_0 = F(x_i + \bar{x}_i) = F$, co oznacza, że funkcja f jest niezależna od x_i (przypadkowi temu odpowiada $w_i=0$).

Jeżeli $f_0 \subseteq f_1$, to wprowadzając pomocniczą funkcję f_p taką, że $f_1 = f_0 + f_p$, możemy przeprowadzić następujące przekształcenie:

$$f(X) = x_i f_1 + \bar{x}_i f_0 = x_i (f_0 + f_p) + \bar{x}_i f_0 = (x_i + \bar{x}_i) f_0 + x_i f_p = f_0 + x_i f_p$$

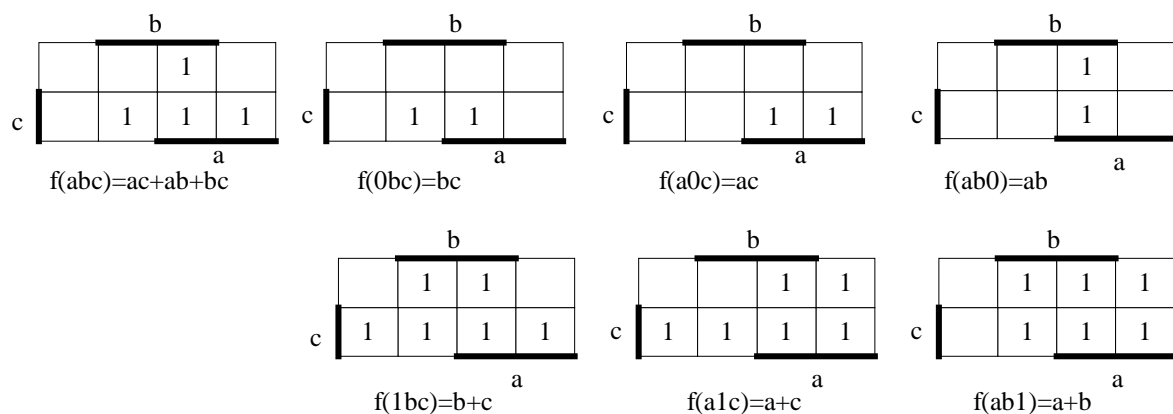
Pokazaliśmy zatem, że gdy $w_i > 0$, funkcja f jest dodatnia ze względu na zmienną x_i .

c.n.p.

Jednorodność jest najprostszym testem sprawdzającym, czy zadana funkcja jest progowa.

Okazuje się jednak, że jest to warunek konieczny i wystarczający realizowalności funkcji za pomocą elementu progowego, jeżeli liczba zmiennych $n \leq 3$.

Silniejszym testem jest własność k -monotoniczności (*k-monotonicity*). Aby zdefiniować tę własność musimy najpierw przypomnieć pojęcie funkcji porównywalnych. Mówimy, że dwie funkcje $f_1(X)$ oraz $f_2(X)$ są porównywalne, jeżeli $(\forall X)(f_1(X) \leq f_2(X))$ albo $(f_1(X) \geq f_2(X))$. Będziemy porównywać ze sobą funkcje resztowe, czyli funkcje otrzymane z funkcji zadanej poprzez ustalenie wartości wybranych zmiennych. Nie wolno zapominać, że porównywać możemy jedynie funkcje zależne od tych samych zmiennych. Dlatego mówiąc o porównywaniu funkcji resztowych będziemy zakładać, że otrzymaliśmy je ustalając wartości tego samego podzbioru zmiennych.



Rys. 7.12 Porównywanie funkcji skróconych (n-1) zmiennych

Definicja 7.5.3

Funkcja przełączająca jest funkcją k -porównywalną, jeżeli porównywalne są dowolne funkcje otrzymane w drodze ustalenia wartości k zmiennych.

Definicja 7.5.4

Funkcja przełączająca nazywana jest funkcją k -monotoniczną, jeżeli jest ona $1, 2, \dots, k$ -porównywalna.

Łatwo wykazać, że 1-monotoniczność odpowiada jednorodności funkcji.

Przykład:

Niech będzie dana funkcja trzech zmiennych $f(abc) = ac + ab + bc$. Funkcja ta jest funkcją 1-monotoniczną, ponieważ funkcje resztowe $(n-1)$ zmiennych są 1-porównywalne dla wszystkich zmiennych. Wszystkie funkcje resztowe przedstawiliśmy na rys.7.12.

Oczywiście $f(0bc) \subseteq f(1bc)$, $f(a0c) \subseteq f(a1c)$ oraz $f(ab0) \subseteq f(ab1)$. Funkcja ta jest także 2-porównywalna, a tym samym jest 2-monotoniczna, ponieważ porównywalne są wszystkie resztowe funkcje przy ustaleniu dwóch zmiennych (rys.7.13).

Definicja 7.5.5

Funkcję przełączającą n zmiennych nazywamy całkowicie monotoniczną (lub krótko monotoniczną), jeżeli jest ona $(n-1)$ -monotoniczna.

Zauważmy, że funkcja z rys.7.13 jest monotoniczna.

Twierdzenie 7.5.3

Dowolna funkcja progowa jest całkowicie monotoniczna.

Na początku badań nad własnością monotoniczności zakładano („dowodzono” zapewne też), że monotoniczność jest warunkiem wystarczającym i koniecznym realizowalności funkcji za pomocą elementu progowego. Okazało się jednak, że Gabelman [2] podał funkcję dziewięciu zmiennych, która jest monotoniczna, lecz nie jest progowa. Muroga [2] formalnie dowiódł, że całkowita monotoniczność funkcji jest wystarczającym warunkiem jedynie dla funkcji $n \leq 8$ zmiennych. Własność 2-monotoniczności jest warunkiem koniecznym i wystarczającym realizowalności funkcji $n \leq 5$ zmiennych.

<div><div><div></div><div></div><div>1</div><div></div></div><div><div>1</div><div>1</div><div>1</div></div></div> <div>$f(abc)=ac+ab+bc$</div>	<div><div><div>0</div><div>0</div><div>0</div><div>0</div></div><div><div>0</div><div>0</div><div>0</div><div>0</div></div></div> <div>$f(00c)=0$</div>	<div><div><div>0</div><div>0</div><div>0</div><div>0</div></div><div><div>0</div><div>0</div><div>0</div><div>0</div></div></div> <div>$f(0b0)=0$</div>	<div><div><div>0</div><div>0</div><div>0</div><div>0</div></div><div><div>0</div><div>0</div><div>0</div><div>0</div></div></div> <div>$f(a00)=0$</div>
<div><div><div></div><div></div><div></div><div></div></div><div><div>1</div><div>1</div><div>1</div><div>1</div></div></div> <div>$f(01c)=f(10c)=c$</div>	<div><div><div></div><div></div><div></div><div></div></div><div><div>1</div><div>1</div><div></div><div></div></div></div> <div>$f(0b1)=f(1b0)=b$</div>	<div><div><div></div><div></div><div></div><div></div></div><div><div></div><div></div><div>1</div><div>1</div></div></div> <div>$f(a01)=f(a10)=a$</div>	
<div><div><div>1</div><div>1</div><div>1</div><div>1</div></div><div><div>1</div><div>1</div><div>1</div><div>1</div></div></div> <div>$f(11c)=1$</div>	<div><div><div>1</div><div>1</div><div>1</div><div>1</div></div><div><div>1</div><div>1</div><div>1</div><div>1</div></div></div> <div>$f(1b1)=1$</div>	<div><div><div>1</div><div>1</div><div>1</div><div>1</div></div><div><div>1</div><div>1</div><div>1</div><div>1</div></div></div> <div>$f(a11)=1$</div>	

Rys. 7.13 Porównywanie funkcji skróconych $(n-2)$ zmiennych

Istnieje silniejsza własność, której spełnienie daje warunek wystarczający i konieczny realizowalności funkcji za pomocą pojedynczego elementu progowego.

Niech będzie dany k wierzchołków przestrzeni n -wymiarowej, dla których $f(X)=1$, oraz k wierzchołków takich, że $f(X)=0$ ($2 \leq k \leq 2^{n-1}$). Jeżeli dla dowolnych takich zbiorów wierzchołków (nie muszą być różne) ich sumy wektorowe są sobie równe, dana funkcja $f(X)$ nie jest realizowalna za pomocą pojedynczego elementu progowego. Mówimy wtedy, że funkcja jest k -sumowalna. Jeżeli nie możemy znaleźć powyższych zbiorów, mówimy, że funkcja jest k -niesumowalna.

Przykład:

Niech będzie dana funkcja przełączająca $f(abc) = \bar{a}b + a\bar{b}$. Wszystkie wierzchołki, dla których $f(abc)=1$, należą do zbioru $\{010, 011, 100, 101\}$, a gdy $f(abc)=0$, do zbioru $\{000, 001, 110, 111\}$. Funkcja jest 2-sumowalna, ponieważ z jednego zbioru możemy wybrać dwa wektory $\{011, 101\}$, a z drugiego $\{001, 111\}$, dla których suma wektorowa wynosi (112) .

Tabela 7.2

Parametry Chow

Liczba zmiennych	$ b_i $	$ a_i $
$n \leq 3$	8 0 0 0	1 0 0 0
	6 2 2 2	2 1 1 1
	4 4 4 0	1 1 1 0
$n \leq 4$	16 0 0 0 0	1 0 0 0 0
	14 2 2 2 2	3 1 1 1 1
	12 4 4 4 0	2 1 1 1 0
	10 6 6 2 2	3 2 2 1 1
	8 8 8 0 0	1 1 1 0 0
	8 8 4 4 4	2 2 1 1 1
	6 6 6 6 6	1 1 1 1 1

Twierdzenie 7.5.4

Funkcja przełączająca jest funkcją progową wtedy i tylko wtedy, gdy funkcja ta jest k -niesumowalna ($2 \leq k \leq 2^{n-1}$).

Na podstawie powyższego łatwo zauważyć, że wykazanie własności niesumowalności jest bardzo pracochłonne.

Dziedzina zajmująca się funkcjami i elementami progowymi nazywana jest logiką progową (*threshold logic*). W logice progowej wyróżnia się dwa ważne zagadnienia:

- badanie, czy dana funkcja jest progowa (synteza elementu progowego),
- realizowanie nieprogowych funkcji przełączających za pomocą elementów progowych (synteza sieci progowych).

Zagadnieniami powyższymi nie będziemy się zajmować, odsyłając Czytelnika do literatury [2][3]. Aby jednak w celach ćwiczeniowych Czytelnik mógł rozpoznawać funkcje progowe (dla $n \leq 4$), proponujemy jedną z najprostszych metod (metoda Chow). Polega ona na obliczaniu parametrów b_i , $i = 0, 1, 2, \dots, n$ według następujących wzorów:

$$b_0 = 2 \cdot \sum f - 2^n$$

$$b_i = 4 \cdot \sum f x_i - 2^n - b_0, 1 \leq i \leq n$$

gdzie: $\sum f$ - liczba jedynek funkcji f w tablicy Karnaugh,

$\sum f x_i$ - liczba jedynek funkcji f w polu obejmowanym przez zmienną x_i na tablicy Karnaugh. Operatory są operatorami arytmetycznymi.

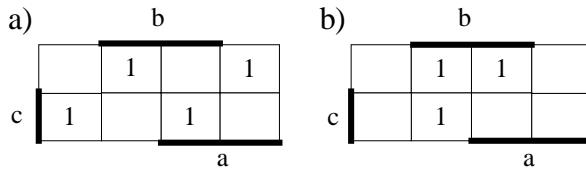
Następnie porządkujemy bezwzględne wartości współczynników b_i w ciąg nierosnący. Porównujemy go z ciągami $|b_i|$ podanymi w tablicach parametrów Chow. Jeżeli uporządkowany ciąg obliczonych wartości $|b_i|$ jest identyczny z podanym w tablicy, oznacza to, że dana funkcja przełączająca jest funkcją progową. Wtedy z kolumny $|a_i|$ tablicy parametrów Chow odczytujemy bezwzględne wartości wag $a_0, a_1, a_2, \dots, a_n$ oraz wartość progu wyliczamy ze wzoru:

$$T = \frac{1}{2} \left(\sum_{i=1}^n a_i - a_0 \right) \quad (7.1)$$

Jeżeli obliczone wartości parametrów Chow nie mają swojego odpowiednika w tablicy, dana funkcja nie jest progowa.

Przykład:

Niech będą dane funkcje przełączające z rys.7.14. Parametry Chow obliczyliśmy dla funkcji z rys. 7.14a i b:



f_1 : $b_0=0$, $b_1=8-8-0=0$, $b_2=0$, $b_3=0$, $|b_i|=(0, 0, 0, 0)$ - funkcja nie jest progowa,

f_2 : $b_0=-2$, $b_1=4-8-(-2)=-2$, $b_2=12-8+2=6$, $b_3=-2$, $|b_i|=(6,2,2,2)$, $|a_i|=(2,1,1,1)$ - funkcja jest progowa.

Rys. 7.14 Przykładowe funkcje przełączające:
a - funkcja nieprogowa, b - funkcja progowa

Wartości wag dla funkcji progowej f_2 obliczamy w następujący sposób: współczynnikowi $b_1=-2$ odpowiada wartość

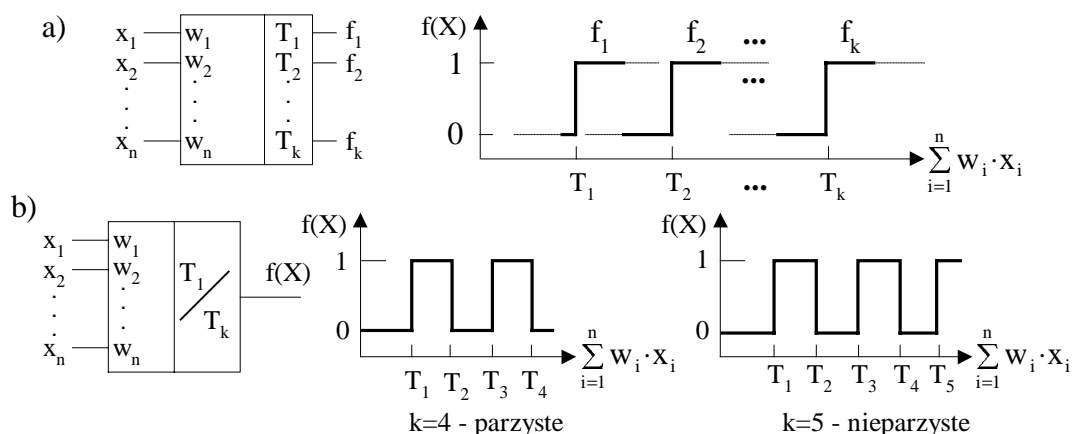
$a_1=-1$ (znak wartości parametru przechodzi na wartość wagi), czyli $w_1=-1$, $b_2=6$ odpowiada $a_2=2$, czyli $w_2=2$; postępując analogicznie otrzymujemy $w_3=-1$. Ze wzoru 7.1 otrzymujemy wartość progę wynoszącą:

$$T = \frac{1}{2} (-1 + 2 - 1 - (-1)) = 0.5$$

Zgodnie z definicją funkcji progowej, możemy przyjąć wartość progę $T=1$ (Dlaczego?).

7.6 Funkcje wieloprogowe

Funkcje progowe po raz pierwszy zastosowano do modelowania komórek nerwowych. Duże znaczenie mają także w teorii rozpoznawania obrazów oraz układach uczących się. Podejmowano i zapewne nadal podejmuje się próby wykorzystania funkcji progowych w układach cyfrowych.



Rys. 7.15 Elementy wieloprogowe: a - wielowyjściowy, b - jednowyjściowy

Można uznać, że obecnie nie ma żadnych przeszkód technologicznych, umożliwiających zbudowanie elementu progowego. Dużym utrudnieniem w szerszym zastosowaniu elementu progowego jest brak bardzo prostych, konkurencyjnych w stosunku do układów konwencjonalnych, metod syntezy i analizy. Metody istniejące są zupełnie odmienne od stosowanych w

układach przełączających. Wiadomo także, że stosunek liczby funkcji progowych do liczby wszystkich funkcji przełączających maleje bardzo szybko ze wzrostem liczby zmiennych. Podjęto zatem próbę uogólnienia funkcji progowej do funkcji wieloprogowej. Rozróżniamy dwa rodzaje funkcji wieloprogowej - funkcje jednowyjściową i wielowyjściową. Należy podkreślić, że funkcje te są diametralnie różne.

Wielowyjściowa bramka wieloprogowa, z wyjściami $f_1(X), \dots, f_k(X)$ oraz z ustalonymi wartościami wag (w_1, w_2, \dots, w_n), realizuje na każdym wyjściu następujące funkcje:

$$f_1(x_1, x_2, \dots, x_n) = 1 \Leftrightarrow \sum_{i=1}^n w_i \cdot x_i \geq T_1$$

$$f_2(x_1, x_2, \dots, x_n) = 1 \Leftrightarrow \sum_{i=1}^n w_i \cdot x_i \geq T_2$$

...

$$f_k(x_1, x_2, \dots, x_n) = 1 \Leftrightarrow \sum_{i=1}^n w_i \cdot x_i \geq T_k$$

gdzie: T_1, \dots, T_k - progi przypisane do danego wyjścia. Przyjmujemy nadal, że wartości wag i progów są liczbami całkowitymi.

Jednowyjściowy element wieloprogowy z wyjściem $f(X)$ zdefiniujemy w następujący sposób:

dla parzystego k :

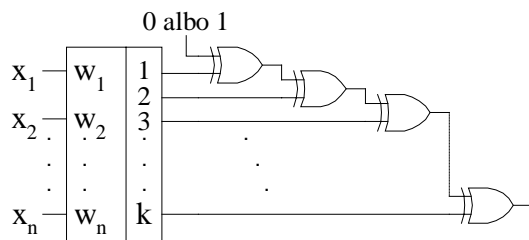
$$D_1 = \bigcup_{j=1}^{k/2} \left\{ (x_1, x_2, \dots, x_n) \mid T_{2j-1} \leq \sum_{i=1}^n w_i \cdot x_i < T_{2j} \right\}$$

dla nieparzystego k :

$$D_1 = \bigcup_{j=1}^{(k-1)/2} \left\{ (x_1, x_2, \dots, x_n) \mid T_{2j-1} \leq \sum_{i=1}^n w_i \cdot x_i < T_{2j} \right\} \cup \left\{ (x_1, x_2, \dots, x_n) \mid T_k \leq \sum_{i=1}^n w_i \cdot x_i \right\}$$

Funkcje wieloprogowe przedstawiono na rys.7.15.

Łatwo zauważyć, że element wielowyjściowy jest prostym rozszerzeniem elementu progowego, ponieważ dla jednego ustalonego zestawu wartości wag mamy w tym elemencie możliwość realizacji kilku funkcji progowych.



Rys. 7.16 Przykład realizacji jednowyjściowego elementu wieloprogowego

O wiele bardziej interesującym elementem jest jednowyjściowy element wieloprogowy, którym interesowali się projektanci układów VLSI (*Very Large Scale Integration*). Zaletą tego elementu jest możliwość realizacji dowolnej funkcji przełączającej, ponieważ każda funkcja przełączająca jest funkcją k -progową, tzn. można ją zrealizować na k -progowym elemencie jednowyjściowym (oczywiście przy wybranym $k \leq 2^n$).

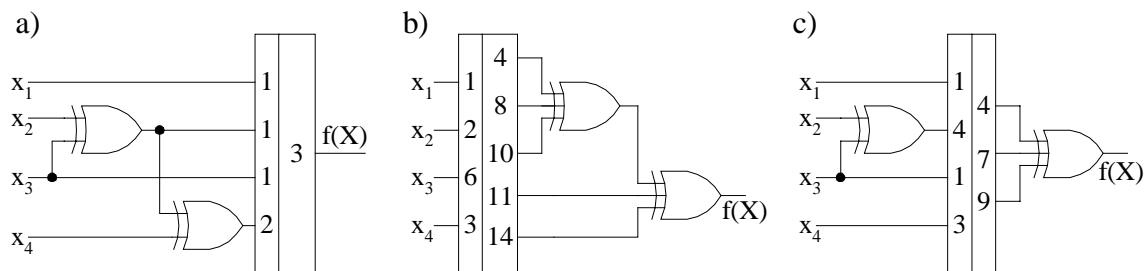
Przykładową realizację jednowyjściowego elementu wieloprogowego przedstawiliśmy na rys. 7.16.

Literatura

- [1] Post E.L., Two-valued iterative systems of mathematical logic, Ann. of Math. Studies, vol. 5, Princeton University Press, 1941
- [2] Muroga S., Threshold logic and its applications, Wiley Interscience, New York, 1971
- [3] Hurst S.L., The logical processing of digital systems, Crane Russak, Inc., New York, 1979

Ćwiczenia

- Wykaż, że własność jednorodności jest równoważna własności 1-monotoniczności.
- Niech $\#$ będzie trójwejściowym operatorem większościowym zdefiniowanym w p.7.4. Czy mają sens następujące napisy?
 - $(p \# q) \# r$
 - $p \# (q \# r)$
- Wykaż, że prawdziwe są następujące związki:
 - $x_1 \# x_1 \# x_2 = x_1$
 - $x_1 \# x_2 \# (x_1 \# x_2 \# x_3) = x_1 \# x_2 \# x_3$
 - $x_1 \# \bar{x}_1 \# x_2 = x_2$
 - $x_1 \# x_2 \# (\bar{x}_1 \# \bar{x}_2 \# x_3) = x_1 \# x_2 \# x_3$
 - $\overline{x_1 \# x_2 \# x_3} = \bar{x}_1 \# \bar{x}_2 \# \bar{x}_3$
- Podaj przykład funkcji monotonicznej, która jednocześnie będzie funkcją liniową.
- Podaj przykład funkcji samodwoistej, która jednocześnie będzie funkcją liniową.
- Pokaż, że przy superpozycji (podstawieniu funkcji w miejsce jej zmiennych) funkcji liniowych otrzymujemy funkcje liniowe.
- Określ liczbę funkcji samodwoistych zależnych od n zmiennych.
- Czy funkcja $x_1x_2 + x_1x_3 + x_2x_3$ i NOT stanowi system funkcjonalnie pełny?
- Dana jest funkcja samodwoista $f = ab + af_1 + bf_2$, gdzie f_1 i f_2 są funkcjami zależnymi od zmiennych c, d, e, \dots . Zazwyczaj studenci twierdzą, że $f_1 = f_2$. Udowodnij to przypuszczenie, bądź wskaż inne rozwiązanie([2]).
- Wykazać, że funkcja f jest funkcją dodatnią ze względu na zmienną x_i wtedy i tylko wtedy, gdy $f(\dots, x_{i-1}, 0, \dots) \cdot \bar{f}(\dots, x_{i-1}, 1, \dots) = 0$.
- Pokaż, że jeżeli $f(X)$ i $g(X)$ są funkcjami przełączającymi, to $g(X) \subset f(X) \Rightarrow g(\bar{X}) \subset \bar{f}(\bar{X})$.
- Podaj przykład SFP zawierającego cztery funkcje, takiego, że jego dowolny podzbiór składający się z trzech funkcji nie jest SFP.
- Jeżeli $f(X): [W; T]$ wykazać, że:
 - $\bar{f}(X): [-W; 1 - T]$,
 - $f(\bar{X}): [-W; T - \sum w_i]$,
 - $\bar{f}(\bar{X}): [W; 1 - T + \sum w_i]$
- Niech będą dane dwie funkcje progowe o strukturach $f_1: [W; T_1]$ oraz $f_2: [W; T_2]$. Wykazać, że gdy $T_1 \geq T_2$, wtedy $f_1 \subseteq f_2$.
- Element progowy o strukturze $[W; T]$ realizuje funkcje f . Znajdź strukturę elementu progowego realizującego funkcje: $f(\dots, x_{i-1}, 0, \dots)$ oraz $f(\dots, x_{i-1}, 1, \dots)$.
- Wykazać, że jeśli $f: [w_1, \dots, w_n; T]$, to



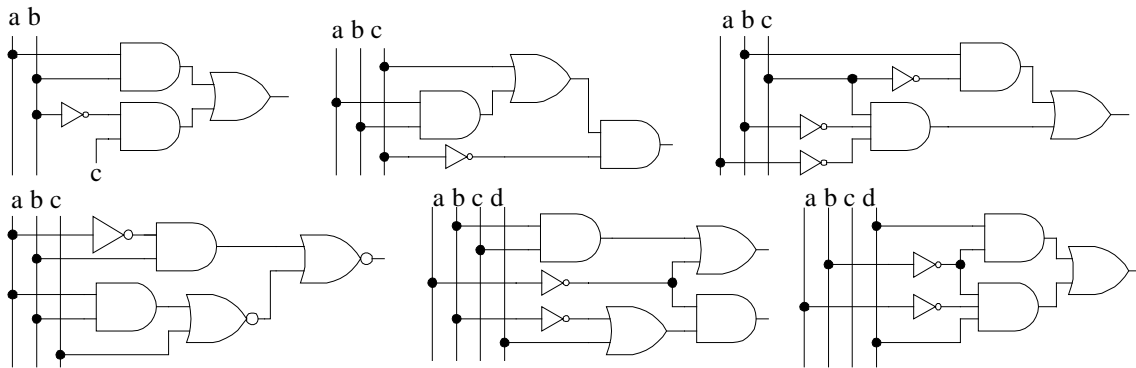
Rys. 7.17 Przykładowe układy wykorzystujące elementy wieloprogowe

a) $f \subseteq \bar{f}(\bar{X})$, gdy $T > \frac{1}{2}(w_1 + \dots + w_n + 1)$

b) $\bar{f}(\bar{X}) \subseteq f$, gdy $T < \frac{1}{2}(w_1 + \dots + w_n + 1)$

c) $\bar{f}(\bar{X}) = f$, gdy $T = \frac{1}{2}(w_1 + \dots + w_n + 1)$

17. Niech będzie dany element progowy z ustalonymi wartościami wag oraz możliwością zmiany wartości progu. Zmieniając wartość progu wykazać, że liczba różnych funkcji realizowanych przez ten element nie będzie większa od 2^n+1 (licząc funkcje stałej 1 i 0).
18. Podaj funkcje realizowane przez układy przedstawione na rys.7.17.
19. Przekształć układy z rys.7.18 w układy NAND tak, by występowała w nich najmniejsza liczba bramek NOT.



Rys. 7.18 Przykładowe układy kombinacyjne

20. Pokaż wszystkie realizacje szesnastu funkcji dwóch zmiennych mając do dyspozycji bramkę ITE i NOT oraz stałe 0 i 1. Bramka ITE (*if-then-else*) realizuje następującą funkcję: $\text{ITE}(f,g,h) = f \cdot g + \bar{f} \cdot h$. Na przykład: $f \cdot g = \text{ITE}(f,g,0)$, $\bar{f} \cdot g = \text{ITE}(f,\bar{g},1)$.

12 ELEMENTY DIAGNOSTYKI UKŁADÓW KOMBINACYJNYCH

12.1 Wprowadzenie

Wyraz diagnoza, w tłumaczeniu z języka greckiego, oznacza rozpoznanie, określenie. W medycynie jest to rozpoznanie stanu pacjenta, w technice - stanu urządzenia technicznego. Przykładami rezultatów diagnozy są zazwyczaj sformułowania: urządzenie sprawne, urządzenie uszkodzone. Diagnostyka jest dziedziną wiedzy zajmującą się teorią i metodami organizacji procesu diagnozy. Jeżeli obiektami diagnozy będą urządzenia techniczne, mówić będziemy o diagnostyce technicznej. W niniejszym rozdziale nasze rozważania będą dotyczyły jedynie układów kombinacyjnych. Jednym z ważniejszych zadań diagnostyki technicznej jest poszukiwanie uszkodzeń, tzn. wskazania miejsca i ewentualnie przyczyn powstania uszkodzenia. Zazwyczaj zależy nam na tym, aby zadanie to było rozwiązane w sposób efektywny, co z kolei wiąże się z organizacją efektywnych algorytmów diagnozy.

Przez pojęcie uszkodzenia układu będziemy rozumieć albo przekroczenie przez pewien parametr układu dopuszczalnej dla niego tolerancji (uszkodzenia parametryczne), albo zniekształcenia logicznej struktury układu (uszkodzenia logiczne).

Wyróżnia się dwa okresy, w których mogą pojawić się uszkodzenia układu cyfrowego: etap produkcji oraz etap eksploatacji układu. Wystąpienie uszkodzenia jest szkodliwe w obu okresach, jednak koszt jego poszukiwania istotnie się różni. Praktyka wskazuje, że jeżeli koszt wykrycia uszkodzenia w układzie scalonym na etapie produkcji przyjmiemy za 1, to koszt poszukiwania uszkodzonego układu scalonego zamontowanego na płycie obwodu drukowanego zwiększa się dziesięciokrotnie. Kolejne 10 razy wzrośnie koszt poszukiwania uszkodzenia, gdy płytka zostanie użyta w systemie cyfrowym. Dalszy dziesięciokrotny wzrost kosztów nastąpi, gdy system cyfrowy przejdzie do normalnej eksploatacji [1]. Wynika z tego prosty wniosek: poszukiwanie uszkodzeń powinno odbywać się na poziomie układu scalonego (w czasie produkcji) lub co najwyżej pakietu. Im wcześniej uszkodzenie zostanie wykryte, tym koszt uruchomienia urządzenia będzie niższy.

Oczywiście może się zdarzyć, że urządzenie ulega uszkodzeniu w trakcie eksploatacji. Przyczyny uszkodzeń są tu inne niż na etapie produkcji, ponieważ zazwyczaj są wynikiem zmęczenia elektrycznego, wilgotności, wahań zasilania i temperatury czy zanieczyszczeń atmosfery. O ile koszt wykrycia uszkodzenia w normalnie eksploatowanym złożonym systemie jest największy, to bardziej groźne są skutki powstania uszkodzenia (błędne działanie programów komputerowych, lawinowe uszkodzanie kolejnych podzespołów, itp.). Jednak najgroźniejsze jest to, że mogą ulec zniszczeniu zgromadzone w systemie banki danych.

W związku z tym zaczęto budować układy cyfrowe tak, by same kontrolowały swoje działanie. Mówimy, że układy takie mają własność samokontroli. Zazwyczaj układ taki po wykryciu nieprawidłowego działania wstrzymuje pracę systemu, nie dopuszczając do rozprzestrzenienia się skutków uszkodzenia. Przykładem takiego podejścia są układy kontroli bitu parzystości we wszelkiego rodzaju pamięciach operacyjnych oraz układach transmisji danych.

Proces poszukiwania uszkodzeń sprowadza się zazwyczaj do poddania badanego układu tzw. testowaniu. Proces ten polega na obserwowaniu reakcji na wyjściach układu na podawaną na jego wejścia sekwencję pobudzeń. Zaobserwowane reakcje porównuje się z reakcjami układu poprawnego. Elementy sekwencji testującej nazywamy testami. Dla zadanego układu kombinacyjnego sekwencją testującą mogą być wszystkie możliwe zestawy wartości zmiennych wejściowych. Podejście takie jest jednak mało efektywne, a nawet niemożliwe do wyko-

nia. Jeśli przyjęlibyśmy, że czas podawania jednego testu na układ wynosi ok. $1[\mu s]$, a zmiennych wejściowych jest 75, to czas, w jakim podalibyśmy wszystkie $2^{75} \approx 3.8 \cdot 10^{22}$ pobudeń, wyniósłby ok. 10^9 lat. Wyliczenie to wskazuje, że należy poszukiwać takiego zbioru testów, którego liczność byłaby dużo mniejsza od 2^n . Zaobserwowano [1], że czas potrzebny do otrzymania zbioru testów zależy od trzeciej potęgi liczby bramek wchodzących w skład układu. Tworząc algorytmy generacji testów musimy zatem poszukiwać “złotego środka”. Uwzględniając fakt, że czas generacji testów zależy od trzeciej potęgi liczby bramek, możemy podzielić układ podlegający testowaniu na dwa mniejsze, zmniejszając tym samym czas generacji ośmiokrotnie (Dlaczego?). Z drugiej strony, musimy zrezygnować z otrzymania minimalnego zbioru testów (tzn. zbioru testów zawierającego minimalną ich liczbę) na rzecz zbioru prawie minimalnego, jeżeli tylko czas generacji testów będzie dużo mniejszy. Dlatego w literaturze istnieje wiele algorytmów generujących zbiory testów, wykrywających wszystkie uszkodzenia układu i takich, że czas zużyty do ich otrzymania nie jest zbyt duży.

12.2 Uszkodzenia układów kombinacyjnych

Błąd (*failure*) jest obserwowalnym objawem niepoprawnego, czyli niezgodnego ze specyfikacją, działania układu. Uszkodzenie (*fault*) to zjawisko zachodzące na poziomie fizycznym układu, które może (ale nie musi) ujawniać się w postaci błędu. Uszkodzenie sprawia, że wartość logiczna w pewnym punkcie układu jest przeciwna założonej. W tym rozdziale będziemy zajmować się trwałymi uszkodzeniami logicznymi (*permanent fault*), których skutek trwa dłużej niż czas trwania procesu testowania.

Uszkodzenie jest zjawiskiem zachodzącym na poziomie fizycznym (tranzystory, diody itp.), zatem poszukiwanie zbioru testów powinno być zależne od technologii wykonania układu, a zadanie to należałoby rozwiązywać na tym właśnie poziomie. Oczywiście byłoby to bardzo uciążliwe. Proces wyznaczania zbioru testów przenosi się zatem na poziom logiczny, ponieważ zaletą produkowanych obecnie układów scalonych jest to, że przeważająca większość uszkodzeń objawia się w postaci logicznej niesprawności układu. Zauważono, że najczęściej występujące uszkodzenia logiczne mogą być reprezentowane za pomocą trzech modeli:

1. sklejenia z wartością logiczną (*stuck-at*),
2. mostkowe (*bridging*) oraz
3. sprzęgające (*stuck-open*).

12.2.1 Uszkodzenia typu sklejenia

Najbardziej rozpowszechnionym modelem uszkodzeń jest model sklejenia z wartością logiczną. Model zakłada, że uszkodzenie objawia się na poziomie bramki jako „sklejenie” jednego z jej wejść lub wyjść (uszkodzenia pojedyncze) ze stałą wartością logiczną 0 (co w skrócie oznaczamy s-a-0) lub wartością logiczną 1 (s-a-1).

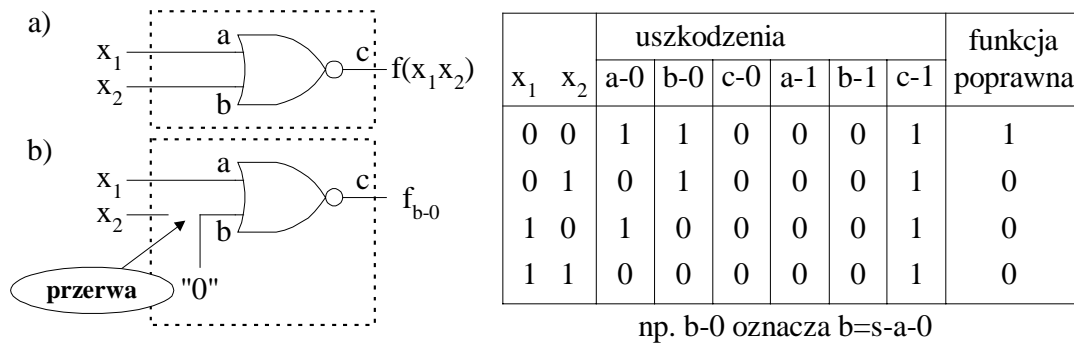
Model ten obejmuje szeroką klasę uszkodzeń typu zwarcie i rozwarcie w wielu rodzajach technologii wykonania bramek (szczególnie TTL).

Rozważmy dwuwejściową bramkę NOR. Wszystkie uszkodzenia bramki NOR możemy zgodnie z modelem przedstawić tak jak to pokazaliśmy na rys.12.1. Przykładowo, kolumna oznaczona b-0 przedstawia wartości funkcji (uszkodzeń) na wyjściu bramki, gdy przewód b „skleimy” ze stałą wartością 0 (rys.12.1b).

Model s-a-{0,1} służy także do reprezentacji wielokrotnych uszkodzeń logicznych. Zakłada się, że wiele linii sygnałowych układu jest sklejonych z 0 lub 1 w tym samym przedziale czasu. Jest to model szczególnie adekwatny do zjawisk zachodzących na etapie

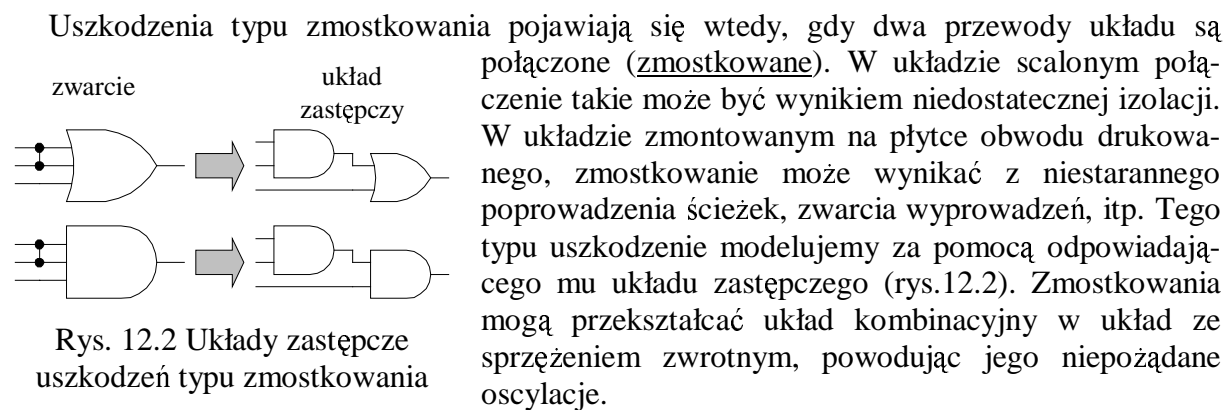
produkcji układu logicznego.

Jeżeli układ znajduje się w normalnej eksploatacji, zazwyczaj zakłada się występowanie uszkodzeń pojedynczych. Oczywiście, założenie to jest słuszne tylko wtedy, gdy okresy między kolejnymi testowaniami układu ujawniają uszkodzenia pojedyncze, które są następnie eliminowane w celu zapobieżenia kolejnym uszkodzeniom (wielokrotnym).



Rys. 12.1 Ilustracja uszkodzeń typu s-a-{0,1}

12.2.2 Uszkodzenia typu zmostkowania



12.2.3 Uszkodzenia sprzęgające

Pewnego typu uszkodzenia są w szczególny sposób uzależnione od technologii, w jakiej wykonano bramki logiczne i nie można ich modelować za pomocą uszkodzenia typu sklejenia. Między innymi, układy wykonane w technologii CMOS (*Complementary Metal Oxide Semiconductor*) ulegają uszkodzeniom, które wprowadzają układ w stan wysokiej impedancji. Ten rodzaj uszkodzeń nazwiemy sprzęgającymi.

W niniejszym podręczniku zajmiemy się jedynie testowaniem układów kombinacyjnych, których uszkodzenia modeluje się za pomocą sklejenia z wartością logiczną.

12.3 Tablica funkcji uszkodzeń

Modelem matematycznym układu kombinacyjnego podlegającego procesowi diagnozy może być tablica funkcji uszkodzeń (rys. 12.3). W praktyce tablica funkcji uszkodzeń nie ma większego znaczenia, ponieważ jej rozmiar jest bardzo duży nawet dla najprostszych układów. Zajmiemy się nią jednak, ponieważ umożliwia pogładowe przedstawienie podstawowych pojęć i algorytmów poszukiwania testów.

Na rys. 12.3 funkcja f oznacza funkcję poprawną, gdy nie ma w układzie żadnego uszkodzenia, f_i funkcję uszkodzonego układu, jeżeli pojawiło się i -te uszkodzenie, a Z_j^i jest

wartością na wyjściu układu, gdy występuje i -te uszkodzenie a na wejście podaliśmy pobudzenie p_j .

Pobudzenie $p_j \in K$ ma własność wykrywalności (*detectability*) dowolnego uszkodzenia ze zbioru S , jeżeli $Z_j \neq Z_j^i$.

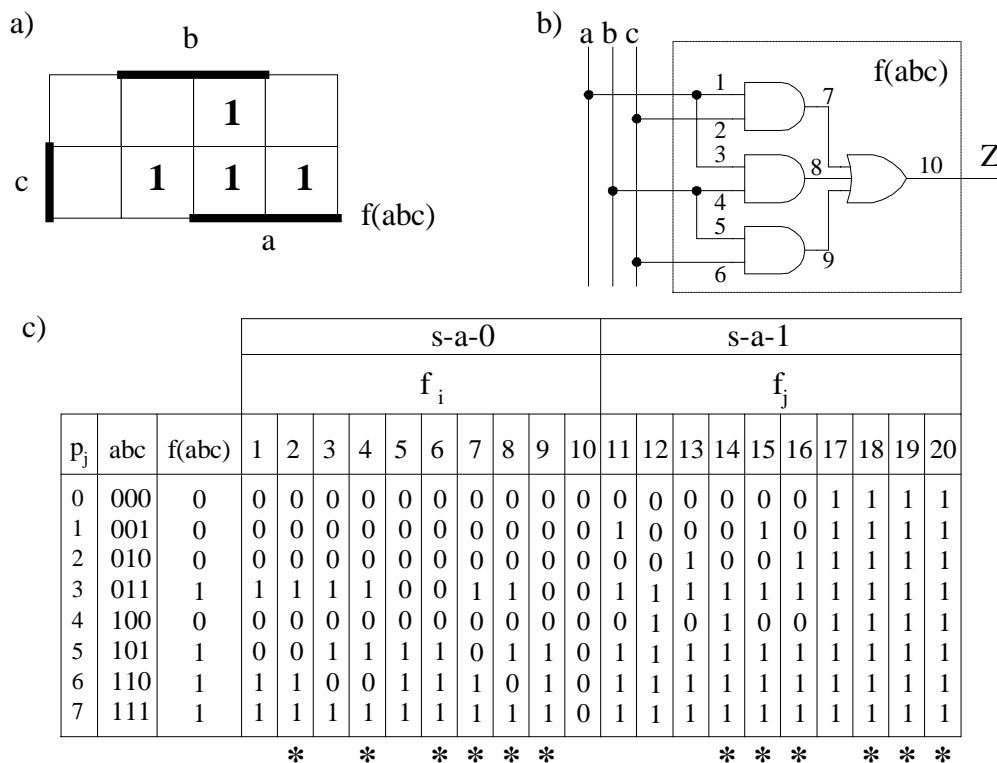
	Pobudzenia	Funkcja poprawna	Funkcje uszkodzeń $\{s\}$				
		f	f_1	...	f_i	...	f_s
{K}	p_1	Z_1	Z_1^1	...	Z_1^i	...	Z_1^s

	p_j	Z_j	Z_j^1	...	Z_j^i	...	Z_j^s

	p_k	Z_k	Z_k^1	...	Z_k^i	...	Z_k^s

Rys. 12.3 Tablica funkcji uszkodzeń

Innymi słowy, i -te uszkodzenie jest wykrywane, jeżeli kolumna reprezentująca funkcję poprawną jest różna od kolumny reprezentującej i -te uszkodzenie. W przeciwnym wypadku mówimy, że uszkodzenie nie jest wykrywane. Pobudzenie $p_j \in K$ ma własność rozdzielania (*distinguishability*) dowolnej pary uszkodzeń $s_i, s_m \in S, i \neq m$, gdy znajdziemy przynajmniej jedno p_j takie, że $Z_j^i \neq Z_j^m$. Zatem własność rozdzielania zachodzi wtedy, gdy wszystkie pary kolumn tablicy, są różne. W przeciwnym wypadku, uszkodzenia nie są rozdzielane (nie są lokalizowane).



Rys. 12.4 Przykładowa tablica funkcji uszkodzeń

Przykład:

Niech będzie dany układ kombinacyjny realizujący funkcję $f(abc)=ac+ab+bc$. Na rys. 12.4c przedstawiliśmy tablice funkcji uszkodzeń, przy założeniu modelu pojedynczych uszkodzeń typu sklejania. Funkcje uszkodzeń f_i , $i=1,2,\dots,10$ odpowiadają sklejaniu i -tego przewodu z wartością zero ($i/0$). Funkcje uszkodzeń f_j , $j=11,12,\dots,20$ odpowiadają sklejaniu i -tego przewodu z wartością jeden ($i/1$), gdzie $i=j-10$. Wnioski, jakie możemy wyciągnąć z przykładowej tablicy funkcji uszkodzeń, są następujące:

1. Dowolne uszkodzenie reprezentowane przez funkcję uszkodzeń f_j , $j=1,2,\dots,20$ jest wykrywalne, ponieważ funkcja poprawna f różni się przynajmniej na jednej pozycji od dowolnej funkcji uszkodzeń. Własność powyższą mają układy nieredundancyjne.
2. Nie wszystkie uszkodzenia mogą być zlokalizowane, ponieważ dla przykładowej pary funkcji uszkodzeń f_3 i f_4 nie ma pobudzenia p_j rozróżniającego te funkcje. Oznacza to, że dwa różne uszkodzenia fizyczne pojawiają się na obserwowalnym wyjściu Z jako nierozróżnialne funkcje uszkodzeń (obserwujemy jedno uszkodzenie logiczne).
3. Rozmiar tablicy funkcji uszkodzeń możemy zmniejszyć, pozostawiając jedynie reprezentanta jednakowych funkcji uszkodzeń (funkcje, których nie bierzemy pod uwagę zaznaczyliśmy gwiazdką).

Zbiorem testów T_w wykrywającym wszystkie uszkodzenia nazywamy zbiór pobudzeń, wśród których znajduje się przynajmniej jedno pobudzenie $t_j \in T_w$ takie, że $Z_j \neq Z_j^i$, $1 \leq i \leq |S|$.

Zbiorem testów T_d lokalizującym wszystkie uszkodzenia nazywamy zbiór pobudzeń, wśród których znajduje się przynajmniej jedno pobudzenie $t_j \in T_d$ rozróżniające dowolną parę funkcji uszkodzeń, czyli $Z_j^k \neq Z_j^l$, $1 \leq k \neq l \leq |S|$.

Zbiór testów nazywamy pełnym, gdy wykrywa (bądź lokalizuje) wszystkie uszkodzenia.

Zbiór testów nazywamy nienadmiarowym, jeżeli odrzucenie z niego dowolnego testu spowoduje, że zbiór nie będzie pełny. Pełny i nienadmiarowy zbiór testów z minimalną ich liczbą nazywamy minimalnym zbiorem testów.

Poszukiwanie wszystkich minimalnych zbiorów testów wykrywających uszkodzenia sprowadza się do wykonania dwóch zasadniczych kroków:

Krok 1 Dla wszystkich par (f, f_i) , $i=1,\dots,|S|$ należy określić podzbiór pobudzeń, dla których $f \neq f_i$. Tak utworzone zbiory oznaczamy przez P_i (zauważmy, że zbiór P_i tworzą wszystkie testy wykrywające i -te uszkodzenie).

Krok 2 Wykorzystując zbiory P_i , należy otrzymać wszystkie zbiory pobudzeń T takie, by w każdym z nich znalazł się przynajmniej jeden test należący do każdego ze zbiorów P_i .

Poszukiwanie wszystkich minimalnych zbiorów lokalizujących wszystkie uszkodzenia, sprowadza się jedynie do zmiany kroku 1.

Krok 1' Dla wszystkich par (f, f_i) , $i=1,\dots,|S|$ oraz par (f_i, f_k) , $i \neq k$, $i, k=1,\dots,|S|$ należy określić podzbiory pobudzeń takie, że $f \neq f_i$ oraz $f_i \neq f_k$.

Algorytm poszukiwania zbiorów, przedstawiliśmy na rys. 12.5. (Uwaga: Z_j^0 oznacza wartość funkcji poprawnej).

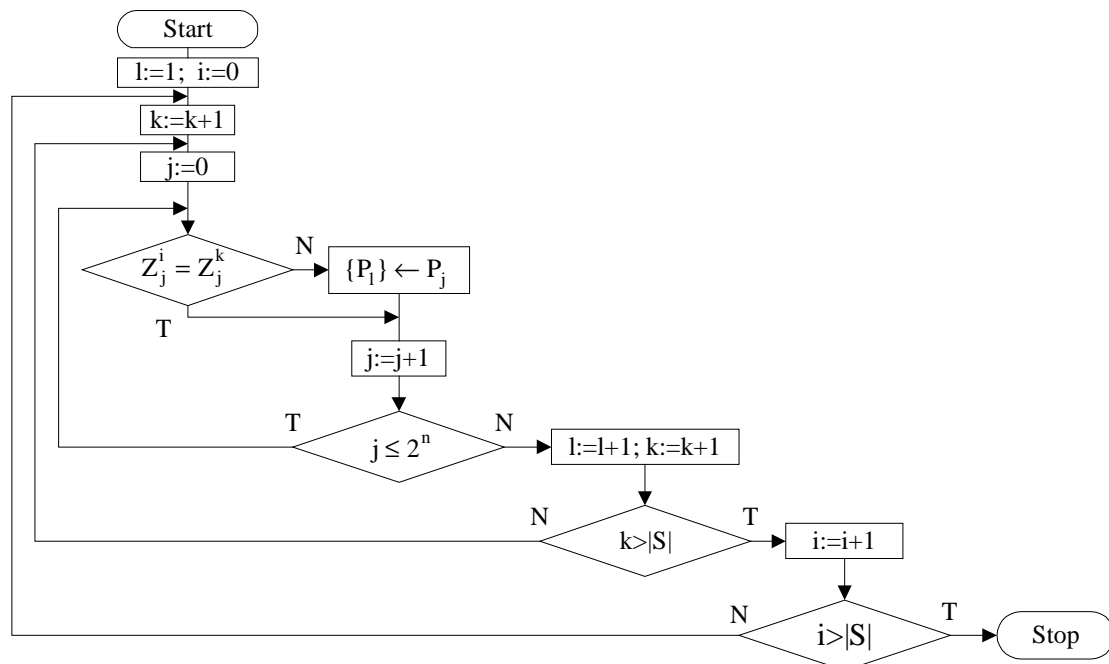
Przykład:

Niech będzie dana tablica funkcji uszkodzeń z rys. 12.4c. Zgodnie z krokiem 1 otrzymamy następujące zbiory P:

$$\begin{array}{lllll} P_1 = \{5\} & P_2 = \{6\} & P_3 = \{3\} & P_4 = \{3, 5, 6, 7\} & P_5 = \{1\} \\ P_6 = \{4\} & P_7 = \{2\} & P_8 = \{0, 1, 2, 4\} & & \end{array}$$

Zbiory P_1 do P_8 wystarczają do uzyskania minimalnych zbiorów testów wykrywających (tzn. możemy przejść do kroku 2).

$$\begin{array}{lll} P_9 = \{5, 6\} & P_{10} = \{3, 5\} & P_{11} = \{3, 6, 7\} & P_{12} = \{1, 5\} \\ P_{13} = \{4, 5\} & P_{14} = \{2, 5\} & P_{15} = \{0, 1, 2, 4, 5\} & P_{16} = \{3, 6\} \\ P_{17} = \{3, 5, 7\} & P_{18} = \{1, 6\} & P_{19} = \{4, 6\} & P_{20} = \{2, 6\} \\ P_{21} = \{0, 1, 2, 4, 6\} & P_{22} = \{5, 6, 7\} & P_{23} = \{1, 3\} & P_{24} = \{3, 4\} \\ P_{25} = \{2, 3\} & P_{26} = \{0, 1, 2, 3, 4\} & P_{27} = \{1, 3, 5, 6, 7\} & P_{28} = \{3, 4, 5, 6, 7\} \\ P_{29} = \{2, 3, 5, 6, 7\} & P_{30} = \{0, 1, 2, 3, 4, 5, 6, 7\} & P_{31} = \{1, 4\} & P_{32} = \{1, 2\} \\ P_{33} = \{0, 2, 4\} & P_{34} = \{2, 4\} & P_{35} = \{0, 1, 2\} & P_{36} = \{0, 1, 4\} \end{array}$$



Rys. 12.5 Algorytm poszukiwania zbiorów P

Formalny sposób przedstawienia kroku 2 jest następujący. Weźmy dwa pierwsze zbiory P_1 i P_2 . Utwórzmy wszystkie możliwe pary wchodzących w ich skład testów. Następnie dokonajmy uproszczenia:

1. każdą parę testów postaci (p, p) , zastępujemy jednym testem (p) ,
2. jeżeli występuje (p) , wykreślamy wszystkie pary postaci $(p, \{q\})$.

Tak otrzymany zbiór par oznaczmy przez T_1 . Z elementów zbioru T_1 i P_3 tworzymy wszystkie możliwe pary. Po uproszczeniu par otrzymujemy zbiór T_2 i algorytm powtarzamy, biorąc zbiór P_4 , itd.

Jeżeli wyczerpiemy wszystkie zbiory P, to podzbiory ostatniego zbioru T są pełnymi i nienadmiarowymi zbiorami testów. Wystarczy wybrać podzbiory zawierające minimalną liczbę testów.

Przykład:

Niech będą dane zbiory P otrzymane w poprzednim przykładzie. Zgodnie z krokiem 2 otrzymamy następujące zbiory T_i (symbol \otimes oznacza operację tworzenia par i ich redukowania):

$$P_1 \otimes P_2 = T_1 = \{5, 6\}$$

$$T_1 \otimes P_3 = T_2 = \{5, 6, 3\}$$

$$T_2 \otimes P_4 = T_3 = \{(5, 6, 3, 3), (5, 6, 3, 5), (5, 6, 3, 6), (5, 6, 3, 7)\} = \{5, 6, 3\}$$

$$T_3 \otimes P_5 = T_4 = \{5, 6, 3, 1\}$$

$$T_4 \otimes P_6 = T_5 = \{5, 6, 3, 1, 4\}$$

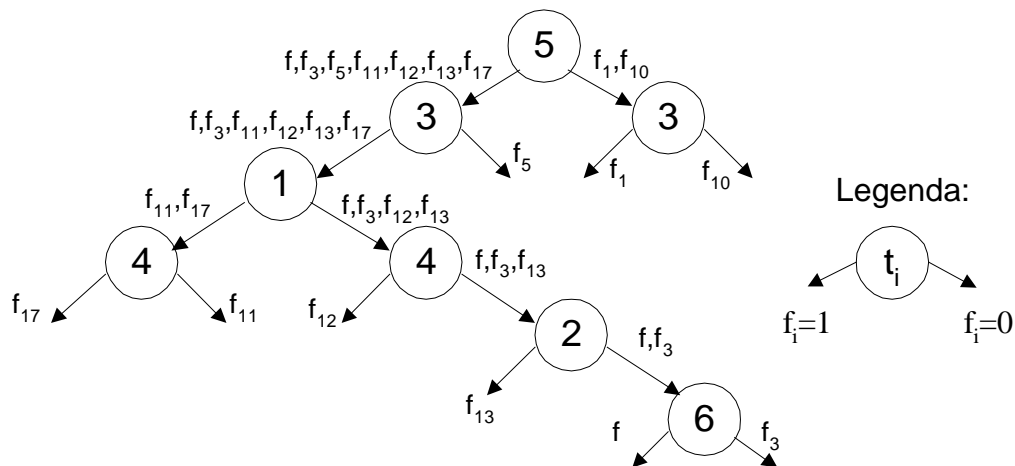
$$T_5 \otimes P_7 = T_6 = \{5, 6, 3, 1, 4, 2\}$$

$$T_6 \otimes P_8 = T_7 = \{5, 6, 3, 1, 4, 2\}$$

Zbiór $T_w = T_7 = \{5, 6, 3, 1, 4, 2\}$ jest zbiorem wykrywającym wszystkie uszkodzenia w układzie z rys.12.4b.

Kontynuując krok 2, to znaczy biorąc pod uwagę zbiory $P_9, P_{10}, \dots, P_{36}$, otrzymamy minimalny zbiór lokalizujący wszystkie uszkodzenia w układzie z rys.12.4b. Dla danego przykładu $T_d = \{5, 6, 3, 1, 4, 2\} = T_w$. Zazwyczaj zbiory lokalizujące uszkodzenia zawierają większą liczbę testów niż zbiór wykrywający uszkodzenia.

Na podstawie zbioru testów wykrywających uszkodzenia można odpowiedzieć na pytanie, czy układ jest uszkodzony. Jeżeli odpowiedź jest pozytywna, zazwyczaj zależy nam na zlokalizowaniu uszkodzenia (jego umiejscowienia). Do tego celu służą zbiory testów lokalizujących T_d .



Rys. 12.6 Drzewo diagnostyczne

Eksperyment diagnostyczny polega na pobudzeniu układu testami lokalizującymi i obserwowaniu wartości na wyjściu układu. Sposób przeprowadzenia eksperymentu przedstawiliśmy na rys. 12.6 w formie drzewa diagnostycznego. Wierzchołkami drzewa są testy wchodzące w skład wyżej otrzymanego zbioru lokalizującego. Gałęzie wychodzące z wierzchołka odpowiadają wynikom poszczególnych testów. Proces lokalizacji ulega zakończeniu, jeżeli dojdziemy do gałązki opisanej pojedynczą funkcją uszkodzeń.

Jeżeli na układ podamy sekwencję testów (5,3,1,4) i otrzymamy w odpowiedzi na wyjściu układu sekwencję postaci (1,1,0,1), to śledząc drzewo diagnostyczne stwierdzamy, że w układzie występuje uszkodzenie reprezentowane przez funkcję f_{12} (uszkodzenie 2/1 albo 4/1, patrz rys. 12.4b). Zauważmy, że w celu lokalizacji uszkodzenia reprezentowanego przez funkcję f_3 należy podać na układ wszystkie testy.

Drzewo diagnostyczne przedstawione na rys.12.6 odpowiada testowaniu kombinacyjnemu (stałemu), ponieważ testy podawane w dowolnej fazie testowania są niezależne od wyników poprzednich testów. W takim przypadku drzewo diagnostyczne jest podzielone na poziomy, w których wszystkie węzły tego samego poziomu reprezentują ten sam test.

Mimo, że liczba testów lokalizujących przy testowaniu kombinacyjnym jest niezależna od kolejności podawania testów, liczba poziomów w drzewie może jednak być zależna od tej

kolejności (podaj przykład). Jeżeli kolejność podawania testów na układ jest uzależniona od wyników poprzednich testów, testowanie nazywamy adaptacyjnym.

12.4 Klasy równoważności uszkodzeń

W układzie kombinacyjnym realizującym funkcję f zbiór testów wykrywający i -te uszkodzenie możemy zdefiniować za pomocą równania $T_i = f \otimes f_i$ oraz zbiór, który wykrywa j -te uszkodzenie za pomocą równania $T_j = f \otimes f_j$. Zbiór testów, który rozróżnia i -te oraz j -te uszkodzenie, może być zdefiniowany przez $f_i \otimes f_j$. Gdy $f_i = f_j$, to żaden test nie rozróżnia tych uszkodzeń. Takie uszkodzenia nazwiemy równoważnymi. Zatem zbiór uszkodzeń jest równoważny, jeżeli dowolny test, który wykrywa jedno z nich, będzie wykrywał wszystkie uszkodzenia, lecz żaden test ich nie rozróżni. Wynika z tego wniosek, iż w celu otrzymania zbioru wykrywającego uszkodzenia wystarczy wziąć pod uwagę tylko jedno uszkodzenie z danej klasy równoważności uszkodzeń.

Metoda wyznaczania klas równoważności uszkodzeń opiera się na pojęciach pochłaniania uszkodzeń (*collapsing*). Najniższy poziom pochłaniania uszkodzeń dotyczy bramek oraz punktów rozłączy (rozgałęzień, *fan-out*).

Wyróżnia się trzy przypadki pochłaniania uszkodzeń:

- PU₁** - pochłanianie uszkodzeń odpowiada istnieniu relacji równoważności między uszkodzeniami związanymi z bramką lub punktem rozłączy.
- PU₂** - pochłanianie uszkodzeń odpowiada istnieniu relacji implikacji między uszkodzeniami pojedynczymi związanymi z bramką lub punktem rozłączy.
- PU₃** - pochłanianie uszkodzeń odpowiada istnieniu relacji równoważności między uszkodzeniami pojedynczymi a grupą uszkodzeń wielokrotnych związanych z bramką lub punktem rozłączy.

Rozważmy bramkę AND o wejściach x, y oraz wyjściu z . Jak nietrudno zauważyć, $x/0 \Leftrightarrow y/0 \Leftrightarrow z/0$, co oznacza $T(x/0)=T(y/0)=T(z/0)=\{(1,1)\}$, gdzie przez $T(x/0)$ oznaczyliśmy zbiór testów wykrywających uszkodzenie $x/0$. Ta klasa równoważności odpowiada pochłanianiu uszkodzeń **PU₁**. Dla tej samej bramki $x/1 \Rightarrow z/1$ oraz $y/1 \Rightarrow z/1$, ponieważ $T(y/1) \subset T(z/1)$ (pochłanianie uszkodzeń **PU₂**). Trzeci przypadek pochłaniania uszkodzeń sprowadza się do zauważenia, że $z/1 \Leftrightarrow (x/1, y/1)$.

Pochłanianie uszkodzeń na poziomie podstawowych bramek jest operacją lokalną, jednak relacje równoważności **PU₁** i **PU₂** są przechodnie. Z tego powodu lokalne stosowanie pochłaniania uszkodzeń równoważnych możemy wykorzystać do wyznaczenia klas uszkodzeń równoważnych i klas uszkodzeń implikowanych w obrębie całego układu. Dlatego znajomość klas uszkodzeń równoważnych jest niezbędna przy otrzymywaniu zbioru testów lokalizujących uszkodzenia, natomiast znajomość klas uszkodzeń implikowanych - przy otrzymywaniu zbioru testów wykrywających uszkodzenia.

Rozważania nad pochłanianiem uszkodzeń doprowadziły do sformułowania ważnego twierdzenia podanego dla klasy uszkodzeń pojedynczych. Przed podaniem twierdzenia wprowadźmy definicję punktu testowania (*checkpoint*):

Definicja 12.4.1

Punktami testowania są wszystkie wejścia pierwotne układu oraz punkty rozgałęzień.

Twierdzenie 12.4.1

Zbiór testów wykrywający uszkodzenia pojedyncze w punktach testowania nieredundancyjnego układu kombinacyjnego jest zbiorem wykrywającym wszystkie uszkodzenia pojedyncze w całym układzie.

Dla układu z rys. 12.4 punktami testowania są punkty 1, 2, 3, 4, 5 oraz 6.

Twierdzenie 12.4.1 ma jedną niedogodność - dotyczy układów nieredundancyjnych, a bardzo często występuje konieczność tworzenia układów redundancyjnych (patrz rozdz. 14.6). Dlatego ważne jest następujące twierdzenie:

Twierdzenie 12.4.2

Zbiór testów wykrywający wszystkie uszkodzenia pojedyncze w nieredundancyjnym układzie kombinacyjnym wykrywa tę samą klasę uszkodzeń w dwupoziomowym układzie redundancyjnym stanowiącym sumę wszystkich implikantów prostych realizowanej funkcji przełączającej, niezależnie od tego, czy w układzie tym wystąpiły dodatkowo uszkodzenia niewykrywalne (w części redundancyjnej).

Twierdzenia dotyczą klasy uszkodzeń pojedynczych. Otrzymywanie zbiorów testów wykrywających uszkodzenia wielokrotne może być znacznie uproszczone, jeżeli weźmiemy pod uwagę kolejne twierdzenie.

Twierdzenie 12.4.3

Dla dowolnego nieredundancyjnego układu kombinacyjnego zbiór testów wykrywający wszystkie uszkodzenia pojedyncze i wielokrotne w punktach testowania układu jest sekwencją wykrywającą wszystkie uszkodzenia pojedyncze i wielokrotne w całym układzie.

Podane twierdzenia w znacznym stopniu upraszczają proces otrzymywania zbiorów testów wykrywających uszkodzenia układów kombinacyjnych. Znacznie uprościmy proces generowania testów, jeżeli narzucimy pewne ograniczenia na strukturę układu, tzn. wykorzystywane bramki i sposób ich połączenia. Biorąc pod uwagę definicję punktów testowania, możemy dojść do wniosku, że gdy projektowane przez nas układy kombinacyjne nie będą zawierać punktów rozgałęzień, wystarczy rozważyć uszkodzenia modelowane na wejściach pierwotnych układu. Układy nie zawierające punktów rozgałęzień nazywamy układami typu drzewa ($n+1 \leq |T| \leq 2\sqrt{n}$).

Prawdopodobne jest, że podczas produkcji pojawią się w układzie uszkodzenia wielokrotne. W praktyce nie ma potrzeby otrzymywania testów dla wszystkich uszkodzeń wielokrotnych. Wystarczy otrzymać zbiór testów wykrywający wszystkie uszkodzenia pojedyncze reprezentowane w punktach testowania, których jest znacznie mniej, a następnie sprawdzić, czy którekolwiek z uszkodzeń wielokrotnych w punktach testowania nie jest wykrywane za pomocą tego zbioru. Następnie dla nie wykrywanych uszkodzeń wielokrotnych wyznaczamy dodatkowe testy.

12.5 Wyznaczanie zbioru testów wykrywających wszystkie uszkodzenia typu sklejenia

Zazwyczaj diagnostyk dysponuje opisem funkcjonalnym układu kombinacyjnego, ale nie zna struktury wewnętrznej układu. Zadanie diagnostyka sprowadza się zatem do wyznaczenia pełnego zbioru testów, który będzie wykrywał wszystkie wykrywalne uszkodzenia w dowolnej realizacji układowej danej funkcji. Ważność tak sformułowanego zadania nie ulega wątpliwości, jeżeli uzmysłowimy sobie fakt, że różni producenci funkcjonalnie jednakowych układów stosują różne struktury oraz technologie wykonania układów scalonych. Dlatego dobrze mieć jeden zbiór testów wykrywający uszkodzenia we wszystkich układach realizujących tę samą funkcję.

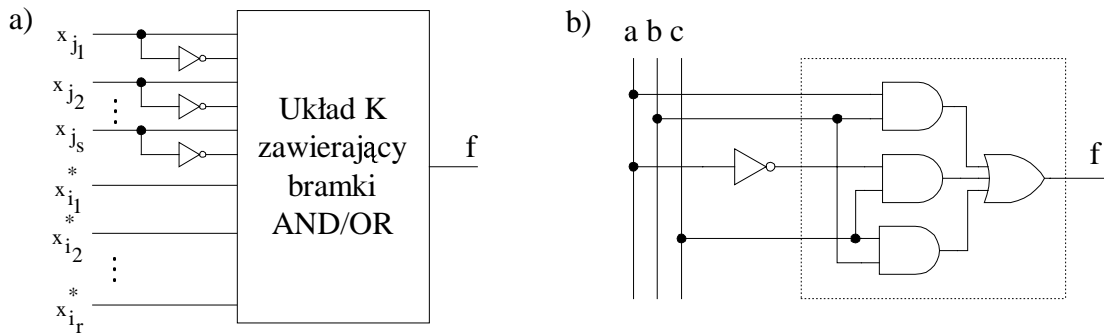
W niniejszym rozdziale przedstawimy metodę wyznaczania uniwersalnego zbioru testów wykrywającego uszkodzenia typu sklejenia (pojedyncze i wielokrotne) w dowolnych układach realizujących zadaną funkcję f.

Definicja 12.5.1

Funkcja f jest jednorodną funkcją dodatnią ze względu na literał x_i^* (tzn. x_i albo \bar{x}_i), jeżeli

$$(\forall X)((f(X) = 1 \wedge x_i^* = 0) \Rightarrow f(Y) = 1)$$

gdzie: Y otrzymano przez dopełnienie x_i^* w X .



Rys. 12.7 Układ uprawniony: a - struktura blokowa, b - przykład

Zauważmy, że gdy f jest dodatnia ze względu na x_i^* , wtedy nie jest dodatnia ze względu na \bar{x}_i^* .

Definicja 12.5.2

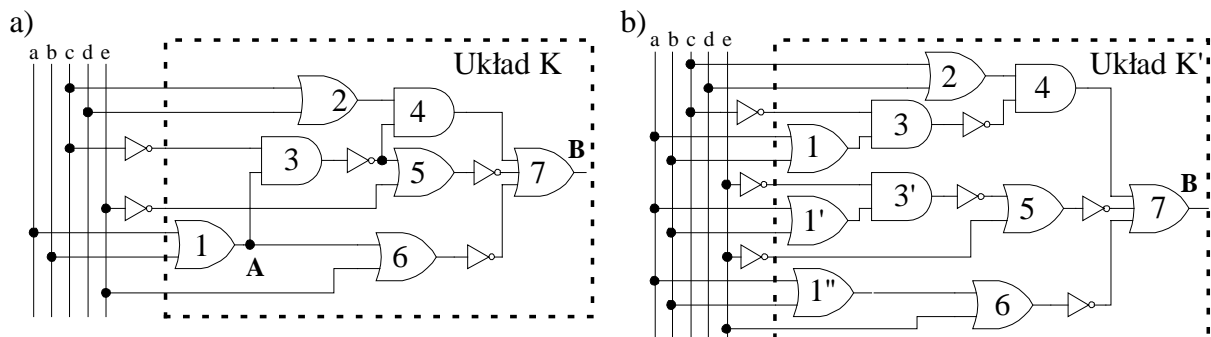
Funkcja f jest jednorodną funkcją dodatnią ze względu na literały $\{x_{i_1}^*, x_{i_2}^*, \dots, x_{i_k}^*\}$, jeżeli jest dodatnia ze względu na $x_{i_j}^*, 1 \leq j \leq k$.

Definicja 12.5.3

Układ kombinacyjny realizujący funkcję f , który zawiera jedynie bramki AND i/lub OR oraz - jeśli są wymagane - bramki NOT jako zewnętrzne wejścia bramek, nazywamy uprawnionym.

Definicja 12.5.4

Jeżeli układ K z rys.12.7a zrealizowany jest tak, że liczba bramek NOT na dowolnej parze ścieżek łączących dwa dowolne punkty układu jest jednakowa, układ nazywamy jednorodnym.



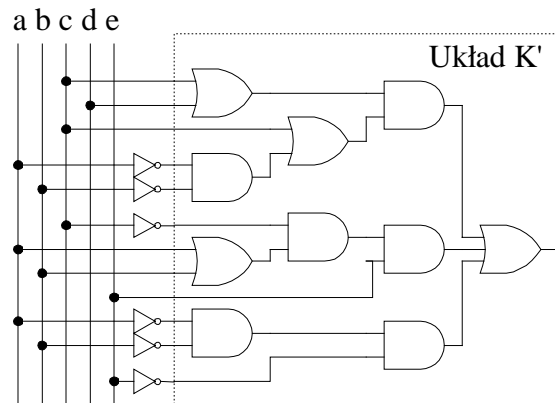
Rys. 12.8 Układy kombinacyjne: a - niejednorodny, b - jednorodny, realizujący funkcję z a)

Na rys.12.8a przedstawiliśmy układ, który nie jest jednorodny, ponieważ w układzie mamy dwie ścieżki łączące punkt A oraz B - jedna prowadząca przez bramki 3,5,7 oraz druga -

przez bramki 6 i 7. Na pierwszej ścieżce znajdują się dwie bramki NOT, na drugiej tylko jedna bramka. Na rys.12.8b przedstawiliśmy modyfikację układu z rys.12.8a, który jest jednorodny i realizuje tę samą funkcję.

Definicja 12.5.5

Uprawniony układ K_u jest równoważny jednorodnemu układowi K_j wtedy i tylko wtedy, gdy zbiór testów T wykrywający wszystkie uszkodzenia typu sklejania w K_u wykrywa także wszystkie uszkodzenia w K_j .



Rys. 12.9 Układ uprawniony realizujący funkcję z rys. 12.8b

Naszym celem będzie wyznaczenie zbioru testów wykrywających wszystkie uszkodzenia w układach uprawnionych i jednorodnych. Na podstawie definicji 12.5.5 o równoważności układów uprawnionych możemy twierdzić, że zbiór testów wyznaczony dla układu uprawnionego wykrywa także uszkodzenia w układzie jednorodnym, jeżeli wykazemy, że dla każdego układu jednorodnego istnieje równoważny układ uprawniony. Podamy zatem sposób, w jaki można przekształcić układ jednorodny w układ uprawniony.

Krok 1. Zastąp bramki NAND (NOR) układem par bramek AND-NOT (OR-NOT), tzn. po bramce AND następuje bramka NOT.

Krok 2. Uporządkuj bramki AND i OR w warstwy. Bramka wejściowa odpowiada warstwie $i=1$ (bramki NOT nie bierzemy pod uwagę przy tworzeniu warstw).

Krok 3. Kolejno występującą po sobie nieparzystą liczbę bramek NOT zamień na jedną bramkę NOT (parzystą ich liczbę potraktuj jako linie bez bramek NOT). Gdy wyjście bramki AND (OR) występującej w i -tej warstwie rozgałęzia się i dochodzi do więcej niż jednego wejścia, wówczas każde rozgałęzienie wychodzące z bramki AND (OR) musi zawierać jedną bramkę NOT albo jej brak, zgodnie z definicją układu jednorodnego. Jeżeli na każdym rozgałęzieniu znajduje się bramka NOT, usuń je i wprowadź jedną bramkę NOT na wyjście rozważanej bramki AND (OR). Jeżeli układ jest uprawniony - stop.

Krok 4. Zastąp każdą parę bramek AND-NOT (OR-NOT) bramką OR(AND) z wejściowymi bramkami NOT (stosując prawo De Morgana).

Krok 5. Jeżeli układ jest uprawniony - stop. W przeciwnym wypadku $i:=i+1$ i przejdź do kroku 3.

Na rys. 12.9 przedstawiliśmy układ uprawniony realizujący funkcję realizowaną przez układ jednorodny z rys.12.8b.

Układy uprawnione będziemy opisywać za pomocą rozszerzonej tablicy prawdy. Jest to tablica, której kolumnami są literały $x_{i_1}^*, x_{i_2}^*, \dots, x_{i_r}^*$ oraz $x_{j_1}, \bar{x}_{j_1}, x_{j_2}, \bar{x}_{j_2}, \dots, x_{j_s}, \bar{x}_{j_s}$. Na rysunku 12.10a,b przedstawiliśmy odpowiednie rozszerzone tablice prawdy dla funkcji $f(abc) = a + b\bar{c}$ i $f(abc) = ac + \bar{b}\bar{c}$. Na rysunku 12.10c podaliśmy rozszerzoną tablicę prawdy dla funkcji realizowanej przez układ z rys. 12.9.

Zestawy wartości literałów występujące w wierszach tablicy prawdy nazwiemy zestawami rozszerzonymi.

Definicja 12.5.6

$$S_1 = \{X \mid f(X) = 0 \wedge (\forall Y)(Y > X \Rightarrow f(Y) = 1)\}$$

$$S_0 = \{X \mid f(X) = 1 \wedge (\forall Y)(Y < X \Rightarrow f(Y) = 0)\}$$

gdzie: X - zestawy rozszerzone.

Dla funkcji opisanych rozszerzonymi tablicami prawdy z rys. 12.10a $S_1 = \{0001, 0110\}$, $S_0 = \{0101, 1001, 1010\}$, a z rys. 12.10b $S_1 = \{00110, 01010, 11001\}$, $S_0 = \{11010, 10110, 00101\}$.

Zbiory S_1 i S_0 mają istotne znaczenie, ponieważ występują w podstawowym dla naszych rozważań twierdzeniu.

Twierdzenie 12.5.1

Zbiór $T = S_1 \cup S_0$ wyznaczony dla funkcji f jest wystarczającym zbiorem testów wykrywających wszystkie uszkodzenia pojedyncze i wielokrotne w dowolnym układzie uprawnionym realizującym funkcję f.

Ze sformułowania twierdzenia wynika, że zbiór T wykrywa także uszkodzenia w dowolnym układzie redundancyjnym realizującym funkcję f. Nietrudno także wykazać, że twierdzenie jest prawdziwe dla układów jednorodnych.

Oczywiście zbiór testów, którymi pobudzimy układ, będzie zawierać jedynie składowe odpowiadające zmiennym afirmowanym rozszerzonej tablicy prawdy. Dla układu realizującego funkcję z rys. 12.10a zbiorem testów będzie $T = \{000, 011, 010, 100, 101\}$.

a)

a	b	c	\bar{c}	f(abc)
0	0	0	1	0
0	0	1	0	0
0	1	0	1	1
0	1	1	0	0
1	0	0	1	1
1	0	1	0	1
1	1	0	1	1
1	1	1	0	1

b)

a	b	\bar{b}	c	\bar{c}	f(abc)
0	0	1	0	1	1
0	0	1	1	0	0
0	1	0	0	1	0
0	1	0	1	0	0
1	0	1	0	1	1
1	0	1	1	0	1
1	1	0	0	1	0
1	1	0	1	0	1

c)

a	\bar{a}	b	\bar{b}	c	\bar{c}	d	e	\bar{e}	f(abcde)
0	1	0	1	0	1	0	0	1	1
0	1	0	1	0	1	0	1	0	0
0	1	0	1	0	1	1	0	1	1
0	1	0	1	0	1	1	1	0	1
0	1	0	1	1	0	0	0	1	1
0	1	0	1	1	0	0	1	0	1
0	1	0	1	1	0	1	0	1	1
0	1	0	1	1	0	1	1	0	1
0	1	1	0	0	1	0	0	1	0
0	1	1	0	0	1	0	1	0	1
0	1	1	0	0	1	1	0	1	0
0	1	1	0	0	1	1	1	0	1
0	1	1	0	1	0	0	0	1	1
0	1	1	0	1	0	0	1	0	1
0	1	1	0	1	0	1	0	1	1
0	1	1	0	1	0	1	1	0	1
1	0	0	1	0	1	0	0	1	0
1	0	0	1	0	1	0	1	0	1
1	0	0	1	0	1	1	0	1	0
1	0	0	1	0	1	1	1	0	1
1	0	0	1	1	0	0	0	1	1
1	0	0	1	1	0	0	1	0	1
1	0	0	1	1	0	1	0	1	1
1	0	0	1	1	0	1	1	0	1
1	0	1	0	0	1	0	0	1	0
1	0	1	0	0	1	0	1	0	1
1	0	1	0	0	1	1	0	1	0
1	0	1	0	0	1	1	1	0	1
1	0	1	0	1	0	0	0	1	1
1	0	1	0	1	0	0	1	0	1
1	0	1	0	1	0	1	0	1	1
1	0	1	0	1	0	1	1	0	1

Rys. 12.10 Rozszerzone tablice prawdy

Przykład:

Rozważmy układ uprawniony na rys.12.11. Rozszerzoną tablicę prawdy przedstawiliśmy na rys. 12.11b. Zbiór $S_0=\{10101,01100,00010\}$, $S_1=\{00011,00101,10100\}$. Zatem zgodnie z twierdzeniem 12.5.1 zbiór $T=\{1011,0110,0000,0001,0011,1010\}$ (opuściliśmy we wszystkich testach czwartą składową). Ponieważ układ z rys. 12.11 jest typu drzewo, więc łatwo się przekonać, że powyższy zbiór testów wykrywa wszystkie uszkodzenia pojedyncze i wielokrotne typu sklejenia. Przy okazji zauważmy, że podzbiór zbioru T postaci $\{1011,0110,0000,0001\}$ wykrywa wszystkie uszkodzenia pojedyncze.

Powyższe wyniki rozważań możemy wykorzystać do wykrywania uszkodzeń w układach specjalnie zaprojektowanych. Otóż, jeżeli dowolną funkcję 2-monotoniczną (patrz rozdz. 7) zrealizujemy w nieredundancyjnym dwupoziomowym układzie AND-OR, to prawdziwe jest następujące twierdzenie:

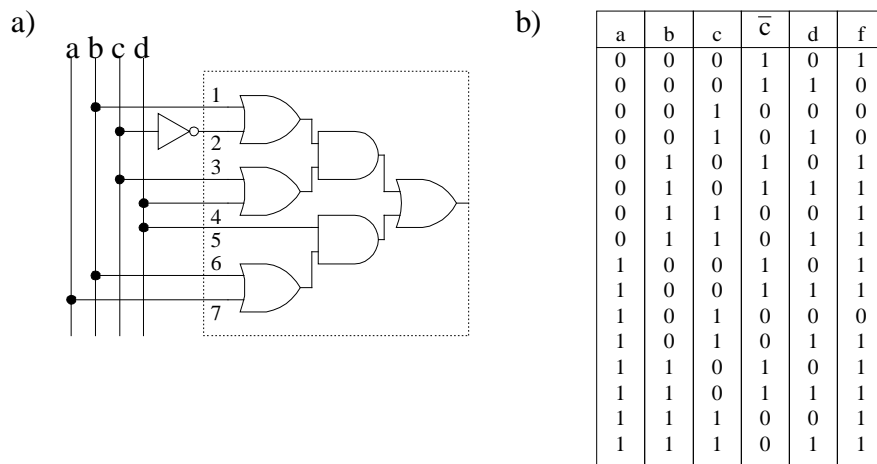
Twierdzenie 12.5.2

Zbiór $T=S_1 \cup S_0$ jest minimalnym zbiorem testów wykrywających wszystkie uszkodzenia typu sklejenia w nieredundancyjnym, dwupoziomowym układzie AND-OR realizującym funkcję 2-monotoniczną.

Przypomnijmy, że dowolna funkcja progowa jest funkcją 2-monotoniczną. Oryginalne uogólnienie uzyskanych wyników do układów logiki wielowartościowej podano w [2]. Zaprezentowany algorytm wyznacza zbiór testów, którego liczność nie przekracza $2n$, gdzie n to liczba zmiennych wejściowych (liczność zbioru jest wprost proporcjonalna do liczby wejść oraz nie zależy od stopnia wartościowości logiki).

Literatura

- [1] Sapieha K., Testowanie i diagnostyka systemów cyfrowych, PWN, Warszawa, 1987
 [2] Kaliś A., Multiple pin fault detection in many valued threshold elements, 1984 Int. Symposium on Multiple-Valued Logic, Winnipeg, Canada, 1984, str. 58-62



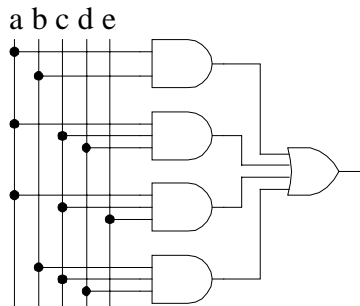
Rys. 12.11 Układ uprawniony: a – realizacja układu, b - rozszerzona tablica prawdy

ĆWICZENIA

1. Wyznacz minimalny zbiór testów wykrywający wszystkie uszkodzenia typu sklejenia w układzie z rys. 12.12 (układ realizuje funkcję progową).
2. Jeżeli układ uprawniony będzie taki, że na wejściach układu K będą doprowadzone zmienne afirmowane oraz ich dopełnienia, to jaka będzie liczność zbioru T otrzymanego

metodą przedstawioną w rozdz. 12.5? Co można powiedzieć o wewnętrznej strukturze układu K?

3. Rozważmy układ z rys. 12.13a. Do poniżej podanych niepełnych zbiorów testów wystarczy dołożyć jeden test, aby otrzymać następujące 4 minimalne zbiory testów: $T1=(5,11, 14,?)$, $T2=(6,11,13,?)$, $T3=(7,9,14,?)$, $T4=(7,10,13,?)$.

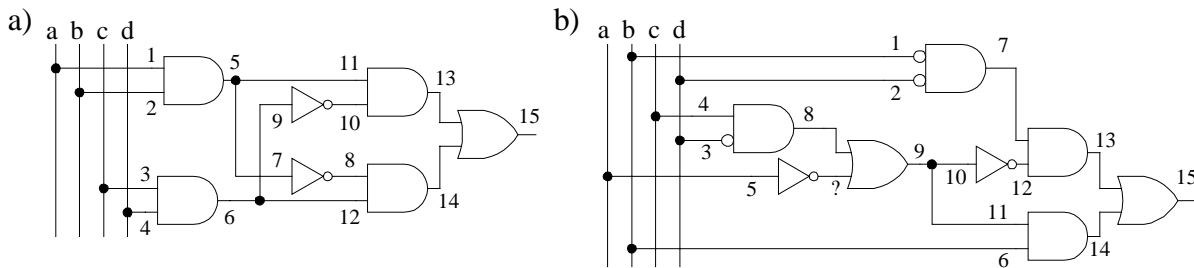


Rys. 12.12 Realizacja funkcji progowej

4. Podaj osiem minimalnych zbiorów testów, wykrywających wszystkie uszkodzenia pojedyncze w układzie z rys. 12.13b.

5. Podaj wszystkie trzy przypadki pochłaniania uszkodzeń dla bramek OR, NOR, NAND, NOT.

6. Uzasadnij, że w celu wygenerowania zbioru testów dla dowolnego układu kombinacyjnego zawierającego tylko bramki NOR, wystarczy wziąć pod uwagę tylko uszkodzenia typu s-a-0.



Rys. 12.13 Przykładowe układy kombinacyjne

14 UKŁADY ASYNCHRONICZNE

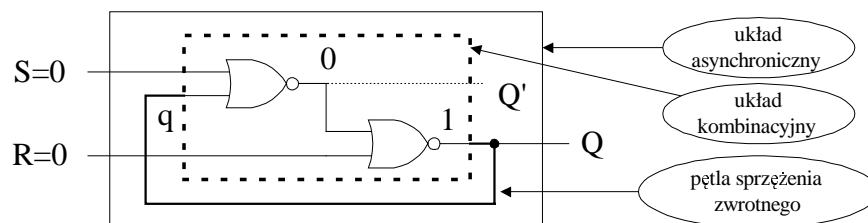
Układy asynchroniczne należą do grupy układów sekwencyjnych. W odróżnieniu od układów kombinacyjnych, układy sekwencyjne nazywamy układami z pamięcią. Dla ustalonych wartości sygnałów wejściowych układ sekwencyjny może wygenerować inną wartość sygnału na swoim wyjściu, gdy te same wartości sygnału wejściowego podamy w innej chwili. Wynika z tego, że wartość wyjścia układu sekwencyjnego zależy nie tylko od aktualnych wartości sygnałów wejściowych, lecz także od sygnałów niosących informacje o wcześniej podanych sygnałach wejściowych (pamiętających historie wejść).

Układy sekwencyjne dzielimy na asynchroniczne i synchroniczne. Jak sama nazwa wskazuje, układ asynchroniczny reaguje bezpośrednio na zmianę sygnału wejściowego ($0 \rightarrow 1$ albo $1 \rightarrow 0$), w przeciwieństwie do układu synchronicznego, w którym reakcja ta zachodzi dopiero po podaniu na układ wyróżnionego sygnału, nazywanego synchronizującym lub zegarowym.

Różnice występujące między układami asynchronicznymi a synchronicznymi wyjaśnimy w rozdziałach następnych.

14.1 Elementarny układ pamiętający

Przed przystąpieniem do omówienia złożonych układów asynchronicznych zajmiemy się przeanalizowaniem prostego układu kombinacyjnego ze sprzężeniem zwrotnym. Układ ten, acz bardzo prosty, jest powszechnie wykorzystywany w układach sekwencyjnych dzięki jego własności pamiętania przez dowolnie długi czas, zadanej wartości logicznej.



Rys. 14.1 Układ kombinacyjny z pętlą sprzężenia zwrotnego

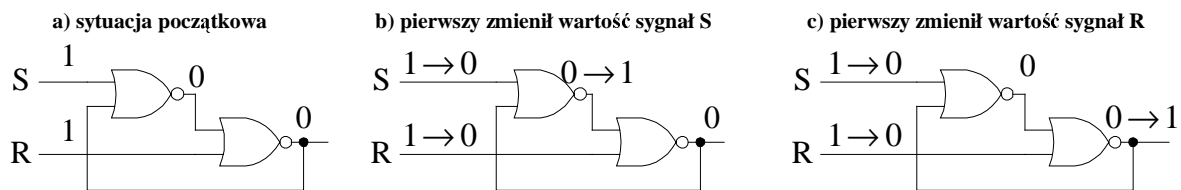
Rozważmy układ z rys. 14.1. Jest to układ kombinacyjny o trzech wejściach S, R, q i jednym wyjściu Q, złożony z dwóch bramek NOR oraz zawierający pętlę sprzężenia zwrotnego. Pętla sprzężenia zwrotnego łączy obserwowalne wyjście Q z wejściem q, umożliwiając propagowanie zmian wartości sygnału wyjściowego na wejście tej samej bramki. Układ kombinacyjny obwiedliśmy linią przerywaną, pętlę sprzężenia pogrubiliśmy, a linie sygnałowe opisałyśmy wartościami logicznymi. Można sprawdzić, że wartości logiczne są poprawne i nie ulegną zmianie, jeżeli tylko nie zmienimy wartości sygnałów S i R. Zastanówmy się zatem, jak będzie się zachowywał układ, gdy wartości tych sygnałów zaczniemy zmieniać, rozpoczynając od $S=0$, $Q=1$, $Q'=0$ oraz $R=0 \rightarrow 1$ (tak będziemy zaznaczać zmianę wartości R z 0 na 1). W odpowiedzi na zmianę wartości sygnału R, sygnał Q przyjmie wartość 0, a Q' wartość 1. Jeżeli nadal będziemy zmieniać wartość R, to wartości Q i Q' nie ulegną zmianie. Załóżmy, że $R=0$, a zmianie ulega $S=0 \rightarrow 1$. Okaze się, że $Q'=0$ i $Q=1$, a kolejne zmiany wartości sygnału S nie spowodują zmiany Q i Q' .

Podsumujmy powyższe zachowanie układu trzema spostrzeżeniami:

1. gdy $S=1$ ($R=1$), to $Q=1$ ($Q=0$),
2. gdy $Q=1$, $R=0$ ($Q=0$, $S=0$), to zmiany wartości S(R) nie zmieniają wartości Q,
3. $Q' = \overline{Q}$.

Układ możemy doprowadzić do sytuacji pokazanej na rys. 14.2a ($S=R=1$, $Q=Q'=0$). Nie byłoby w tym może nic dziwnego (oprócz tego, że $Q=Q'$), gdyby nie wrażliwość układu na kolejność, w jakiej sygnały S i R będą przyjmować wartość 0. Na rys. 14.2b przedstawiliśmy końcową sytuację, gdy jako pierwszy zmienił swą wartość sygnał $S=1 \rightarrow 0$, a jako drugi sygnał $R=1 \rightarrow 0$ (otrzymaliśmy $Q=0$, $Q'=1$), zaś na rys. 14.2c jako pierwszy zmieniliśmy R a następnie S ($Q=1$, $Q'=0$). Zauważmy, że w obu przypadkach rozpoczynaliśmy zmiany od $S=R=1$ i dochodziliśmy do $S=R=0$, lecz Q i Q' różnią się wartościami. Podsumujmy powyższe następującymi spostrzeżeniami:

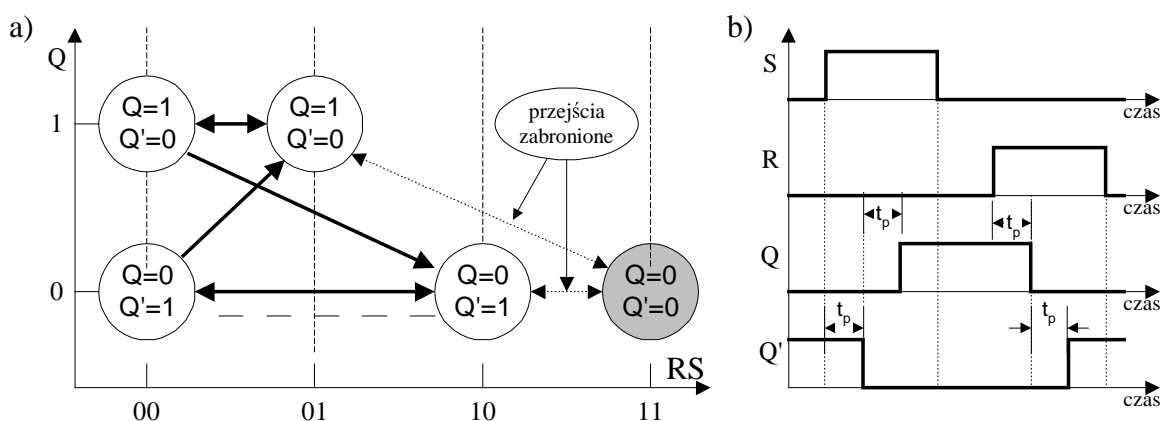
1. gdy $S=R=1$, to $Q=Q'$,
2. gdy $S=R=1$, to wartość sygnału Q zależy od kolejności zmian sygnałów S i R .



Rys. 14.2 Wrażliwość układu na kolejność zmian wartości sygnałów wejściowych

Jeżeli pierwsze spostrzeżenia nasuwały nam pomysł wykorzystania układu do pamiętania wartości 1, (gdy $S=1$, $R=0$) albo wartości 0, (gdy $R=1$, $S=0$), to powyższe zachowanie się układu jest dosyć niepokojące, ponieważ może się zdarzyć, że przy $S=R=0$, $Q=1$ albo $Q=0$.

Chcielibyśmy zapewne, by przy ustalonych wartościach S i R wartość Q była ustalona i zawsze ta sama. Okazuje się, że warunek ten będzie spełniony tylko wtedy, gdy nie dopuścimy by jednocześnie $S=R=1$, zapewniając równość $Q'=\overline{Q}$ (czyli nie musimy stosować dodatkowych bramek NOT, gdy potrzebujemy dopełnienia Q). Istnieje jeszcze jeden powód, dla którego nie należy dopuścić do sytuacji $R=S=1$. Otóż, jeżeli w takim stanie wejść dopuścilibyśmy do prawie jednoczesnej zmiany obu sygnałów z 1 na 0, to na wyjściu Q przerzutnika pojawią się oscylacje (ciągłe zmiany z 1 na 0 i 0 na 1). W rzeczywistym układzie oscylacje przemijają i na wyjściu Q ustali się jedna z wartości 0 albo 1. Problem w tym, że nie możemy przewidzieć, która z nich.



Rys. 14.3 Elementarny układ pamiętający, a - ilustracja działania, b - diagram czasowy

Na rys. 14.3 przedstawiliśmy ilustrację działania układu oraz jego diagram czasowy. Założyliśmy, że czas propagacji każdej z dwóch bramek NOR jest identyczny i wynosi t_p (czas propagacji bramki nazwiemy czas jaki upłynął od chwili zmiany wartości sygnału na wejściu, do chwili zmiany wartości sygnału na wyjściu). Z diagramu możemy wywnioskować, że czas

propagacji zmiany sygnału $S=0 \rightarrow 1$ do wyjścia Q wynosi $2t_p$ (ile wynosi czas propagacji zmiany na wejściu $R=0 \rightarrow 1$ do wyjścia Q ?).

Omawiany układ nazywamy asynchronicznym przerzutnikiem SR (słowo przerzutnik jest tutaj trochę mylące, ponieważ mianem przerzutnika będziemy nazywać inne układy wykorzystywane w układach synchronicznych), albo zatrząskiem SR (*SR latch*).

14.2 Układ kombinacyjny ze sprzężeniem zwrotnym

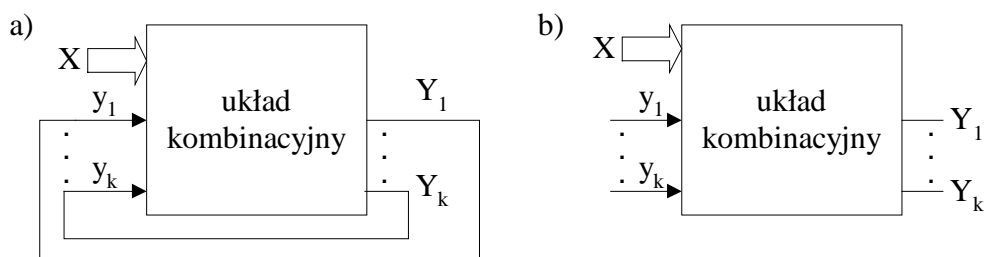
Dowolny układ asynchroniczny możemy przedstawić jako układ kombinacyjny objęty pętlą (pętlami) sprzężenia zwrotnego, co przedstawiliśmy na rys.14.4a. W tym podrozdziale wyjaśnimy pojęcie stanu stabilnego i niestabilnego układu asynchronicznego. W tym celu posłużymy się układem z przerwana pętlą sprzężenia zwrotnego (rys.14.4b), który to układ jest niczym innym, jak zwykłym układem kombinacyjnym. Układ ten potrafimy opisać za pomocą następującego układu funkcji przełączających:

$$Y_1 = f_1(X; y_1, y_2, \dots, y_n)$$

$$Y_2 = f_2(X; y_1, y_2, \dots, y_n)$$

...

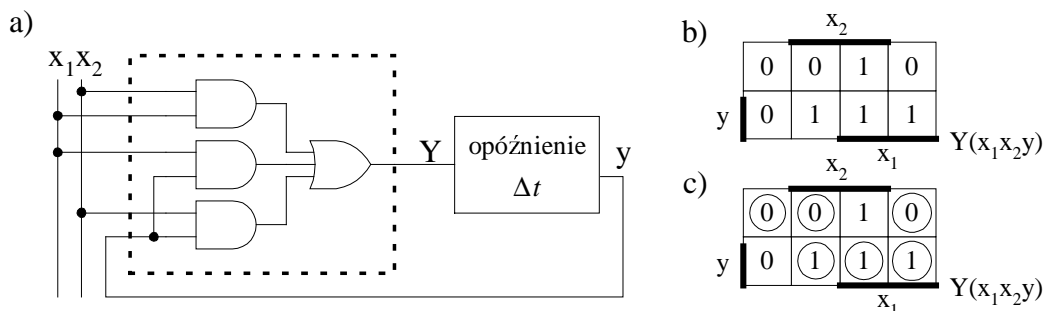
$$Y_k = f_k(X; y_1, y_2, \dots, y_n)$$



Rys. 14.4 Model układu asynchronicznego: a - model ogólny, b - układ z rozwartymi pętlami sprzężenia zwrotnego

Sygnały y_1 do y_k możemy podzielić na dwie następujące klasy (patrz rys. 14.4b):

- a) Y_i jest równy postulowanemu y_i dla $\forall i$,
- b) $Y_i \neq y_i$ dla jednej (bądź wielu) wartości i .



Rys. 14.5 Przykładowa realizacja układu asynchronicznego

Zestaw wartości y_i należące do klasy 1 nazywamy stanami stabilnymi, ponieważ pozostają one nie zmienione aż do odpowiedniej zmiany wartości zmiennej wejściowej (X).

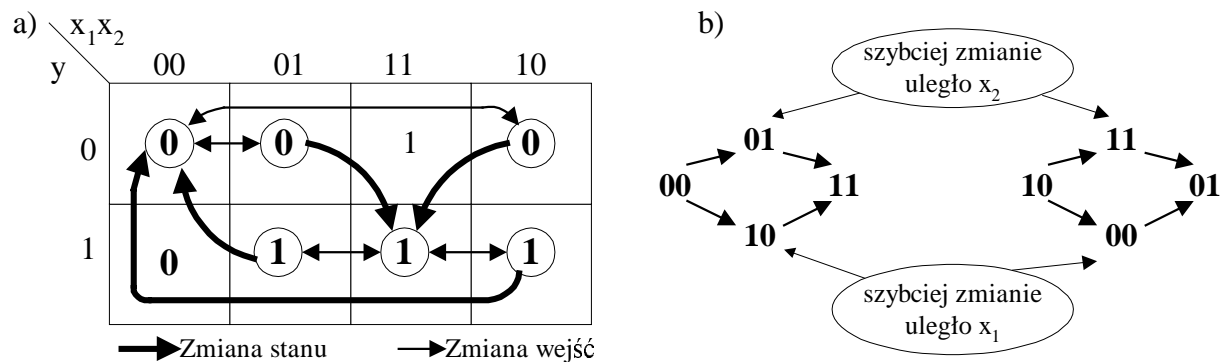
Zestaw wartości y_i z klasy 2 nazywamy stanami niestabilnymi, ponieważ mogą one istnieć jedynie przez czas krótszy od czasu opóźnienia wprowadzonego przez układ kombinacyjny (istotę opóźnień omówimy w p.15.6). Układ tak długo przebywa w stanie niestabilnym, aż

spełniony zostanie warunek definiujący klasę 1, tzn. gdy osiągnie on jakiegokolwiek stanu stabilnego. Innymi słowy, stany niestabilne należy tutaj uważać za kroki pośrednie umożliwiające przejście z bieżącego do następnego stanu stabilnego. Wartości Y nazywamy wzbudzeniem układu.

Rozważmy układ z rys.14.5a. W celu podkreślenia, że każdy rzeczywisty układ kombinacyjny wprowadza pewne opóźnienie zmian sygnałów propagujących się w kierunku wyjścia, na wyjściu układu wprowadzono element opóźniający. Przyjmiemy teraz, że sam układ kombinacyjny żadnego opóźnienia nie wnosi.

Układ z rys. 14.5a można opisać wyrażeniem $Y(x_1x_2y) = x_1y + x_1x_2 + x_2y$ oraz wpisać do tablicy Karnaugh (rys. 14.5 b) .

Wykorzystując tablice Karnaugh określimy wszystkie stany stabilne, zaznaczając je obwódką (wiemy już, że w przypadku osiągnięcia przez układ stanu stabilnego zachodzi równość $Y=y$). W wyniku otrzymamy tablicę pokazaną na rys. 14.5c. Pozostałe nie obwiedzione stany, są stanami niestabilnymi. Tablicę Karnaugh z rys. 14.5c przedstawia się w nieco zmodyfikowanej postaci (patrz rys.14.6), którą nazywamy wtórną tablicą Huffmana. Dzięki tej tablicy można prześledzić zachowanie się układu, bez powoływania się na jego realizację układową (schemat logiczny).



Rys. 14.6a - tablica (wtórna) Huffmana, b - próba wymuszenia "jednoczesnej" zmiany wartości dwóch zmiennych wejściowych

Jeżeli zatem przyjmujemy, że $x_1, x_2=0$ oraz $y=0$, to dla takiego zestawu wartości sygnałów $Y=0$, czyli układ znajduje się w stanie stabilnym (lewy górny róg tablicy z rys. 14.6a). Niech wartość zmiennej wejściowej $x_2 = 0 \rightarrow 1$. W wyniku tej zmiany przesuniemy się do drugiej kolumny tablicy (oznaczonej $x_1x_2=01$), osiągając stan $x_1 = 0, x_2 = 1$ oraz $y=0$. Oczywiście nadal $Y=0$, tzn. układ osiągnął stan stabilny (ten sam co poprzednio). Pozwólmy teraz, aby wartość zmiennej wejściowej x_1 uległa zmianie $0 \rightarrow 1$. Przesuwamy się do kolumny oznaczonej 11, będąc nadal w wierszu $y=0$. Na przecięciu rozważanej kolumny i wiersza $y=0$ znajduje się stan niestabilny (bo $y \neq Y$). Układ pozostanie w tym stanie tylko przez czas τ , przechodząc następnie do stanu stabilnego. Zauważmy, że po czasie τ zmienna y przyjmie wartość 1, co oznacza przesunięcie do drugiego wiersza w tablicy, w którym $y=Y$, a więc osiągnięcie przez układ stanu stabilnego. W trakcie przechodzenia ze stanu niestabilnego do stabilnego nie zmieniamy wartości zmiennych wejściowych x_1 i x_2 .

Na rys. 14.6a zaznaczyliśmy strzałkami możliwe kierunki poruszania się po tablicy Huffmana (dlaczego cienkie strzałki obustronnie zakończone są grotami?). Nie zaznaczyliśmy przejścia między kolumną 00 a 11 oraz 01 a 10, ponieważ nigdy nie wymusimy jednoczesnej zmiany wartości dwóch sygnałów. Zawsze w tym przypadku wygenerowane zostaną pośred-

nie wartości 01 albo 10 (00 albo 11 dla próby $01 \rightarrow 10$), w zależności od tego, która zmienna osiągnie jako pierwsza wartość wyróżnioną (patrz rys. 14.6b).

W celu usystematyzowania wiadomości wypiszmy wprost założenia poczynione, przy analizie układów asynchronicznych:

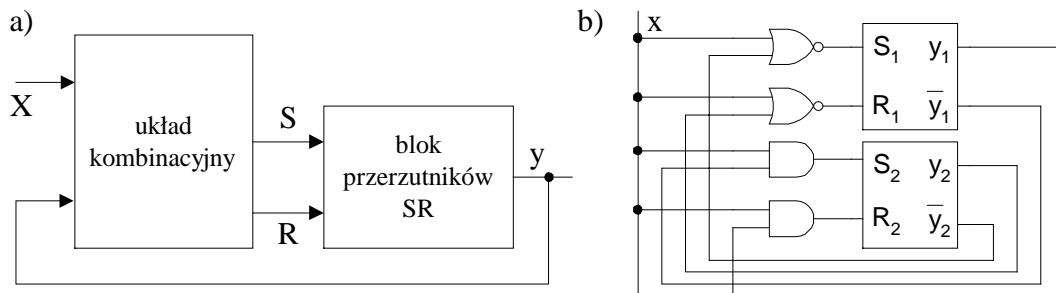
1. dopuszczaliśmy zmianę tylko jednego sygnału wejściowego w danej chwili,
2. zmiana sygnału wejściowego następowała po osiągnięciu stanu stabilnego wymuszonego poprzednią zmianą sygnału wejściowego.

14.3 Układ kombinacyjny z wyodrębnionym blokiem przerzutników asynchronicznych

W tym punkcie przeanalizujemy układ asynchroniczny zawierający w pętli sprzężenia zwrotnego przerzutniki asynchroniczne (rys. 14.7a). Za przykład do analizy posłużmy nam układ przedstawiony na rys. 14.7b. Funkcje wzbudzające poszczególne wejścia asynchronicznych przerzutników SR są następujące:

$$\text{przerzutnik nr 1: } S_1 = \bar{x} \cdot y_2, \quad R_1 = \bar{x} \cdot \bar{y}_2,$$

$$\text{przerzutnik nr 2: } S_2 = x \cdot \bar{y}_1, \quad R_2 = x \cdot y_1.$$



Rys. 14.7 Układ asynchroniczny wykorzystujący przerzutnik asynchroniczny SR: a - struktura blokowa, b - przykładowy układ

Utwórzmy dwie tablice Karnaugh, po jednej dla każdego przerzutnika (rys. 14.8). Wpiszmy do każdej z nich funkcje wzbudzające wejścia S i R rozważanego przerzutnika. Celem odróżnienia od siebie obu funkcji wpisaliśmy obok wartości 1 duże litery S oraz R (wartości funkcji równych zero nie wpisujemy).

Zwróćmy uwagę, że w obu tablicach nie ma kratki, w której wpisaliśmy S oraz R, co oznacza, że działanie przerzutnika asynchronicznego będzie ściśle określone (porównaj p.14.1). Analizę rozpoczniemy od kratki $xy_1y_2=011$ w tablicy opisującej funkcje wzbudzeń przerzutnika nr 1 (rys. 14.8a).

a)

		x	
		0	1
y_1y_2	00	1R	
	01	1S	
	11	1S	
	10	1R	

S_1R_1

wzbudzenie
przerzutnika nr 1

b)

		x	
		0	1
y_1y_2	00		1S
	01		1S
	11		1R
	10		1R

S_2R_2

wzbudzenie
przerzutnika nr 2

c)

		x	
		0	1
y_1y_2	00	r	
	01	S	
	11	s	
	10	R	

S_1R_1

wzbudzenie
przerzutnika nr 1

d)

		x	
		0	1
y_1y_2	00		S
	01		s
	11		R
	10		r

S_2R_2

wzbudzenie
przerzutnika nr 2

Rys. 14.8 Tablice wzbudzeń: a,b - tablice z wpisanymi funkcjami wzbudzeń, c,d – przekształcone tablice wzbudzeń

Na wejście S tego przerzutnika podajemy wartość 1, a stan przerzutnika $y_1=1$. Zgodnie z zasadą działania przerzutnika asynchronicznego, wartość y_1 nie ulegnie zmianie nawet wtedy, gdy na wejście S podamy wartość 0 (czyli wartość sygnału na wejściu S jest obojętna). Spełnienie takich warunków zaznaczamy zmieniając dużą literę S na małe s.

W kratce $xy_1y_2=001$ wpisaliśmy dużą literę S, czyli $S_1=1$ oraz $y_1=0$. Przerzutnik zmieni swój stan $y_1 = 0 \rightarrow 1$, co zaznaczamy pozostawiając w rozważanej kratce dużą literę S. Weźmy pod uwagę kratkę $xy_1y_2=000$. Na wejście R_1 przerzutnika podajemy wartość 1 oraz $y_1=0$. Z zasady działania przerzutnika wiemy, że wartość sygnału na jego wyjściu nie ulegnie zmianie nawet wtedy, gdy na wejście R podamy wartość 0. Oznacza to, że dla $y_1=0$ wartość funkcji wzbudzającej wejście R jest obojętna. Sytuację taką zaznaczamy zamieniając dużą literę R na małe r. Z kolei gdy rozważamy kratkę $xy_1y_2=010$, która odpowiada sytuacji $R_1=1$ oraz $y_1=1$, to wartość na wyjściu przerzutnika ulegnie zmianie ($y_1 = 1 \rightarrow 0$). W kratce tej pozostawimy dużą literę R. W rezultacie otrzymamy tablicę przedstawioną na rys. 14.8c.

Postępując identycznie z tablicą opisującą funkcje wzbudzeń przerzutnika nr 2, otrzymamy tablicę pokazaną na rys. 14.8d.

Podsumowując, zamieniamy $S_i \rightarrow s_i$ ($R_i \rightarrow r_i$) w tych kratkach tablicy, które leżą w polu $y_i=1(0)$, gdzie i oznacza numer przerzutnika.

Nietrudno zauważyć, że odpowiadające sobie kratki w obu tablicach (rys. 14.8c,d) nie zawierające ani R ani S będą reprezentować stan stabilny układu, ponieważ żadna ze zmiennych stanu nie zmienia swojej wartości. Wypełnioną stanami stabilnymi tablicę przedstawia rys. 14.9a.

Wystarczy uzupełnić ją o stany niestabilne, aby otrzymać kompletną tablicę Huffmana. Zasada wpisywania stanów niestabilnych do tablicy jest następująca. Rozważmy kratkę $xy_1y_2=100$, która jest typu S dla przerzutnika nr 2. Oznacza to, że wartość y_2 ulegnie zmianie, czyli otrzymamy $xy_1y_2=101$, co odpowiada stanowi stabilnemu (2). Stąd w rozważanej kratce wpiszemy stan niestabilny 2. Postępując analogicznie z pozostałymi (pustymi) kratkami otrzymamy tablicę przedstawioną na rys. 14.9b.

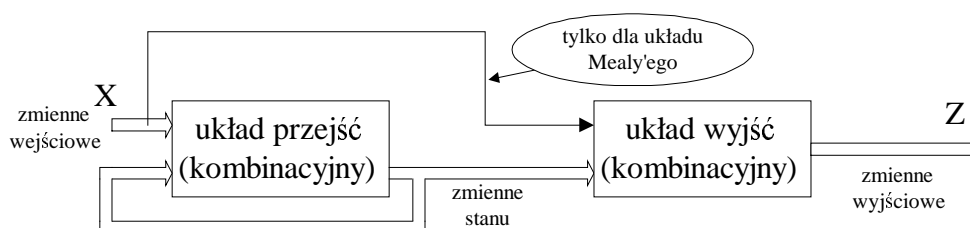
		x			
				y ₁ y ₂	
		0	1		
00		(1)			
01			(2)		
11		(3)			
10			(4)		

		x			
				y ₁ y ₂	
		0	1		
00		(1)	2		
01		3	(2)		
11		(3)	4		
10		1	(4)		

Rys. 14.9 Tablica Huffmana dla układu z rys. 14.7: a - z wpisanymi stanami stabilnymi, b - kompletna tablica Huffmana

14.4 Blokowa struktura układów asynchronicznych

W układach sekwencyjnych wartości wyjść zależą nie tylko od bieżącej wartości wejść, lecz także od ich wartości poprzednich. Zmienne, które „pamiętają” poprzednie sekwencje zmian wartości wejść, nazywamy zmiennymi stanu. Aktualna wartość zmiennych stanu jest zatem prostą konsekwencją historii wejść. Jak już wiemy, zdolnością pamiętania charakteryzują się układy kombinacyjne ze sprzężeniem zwrotnym.

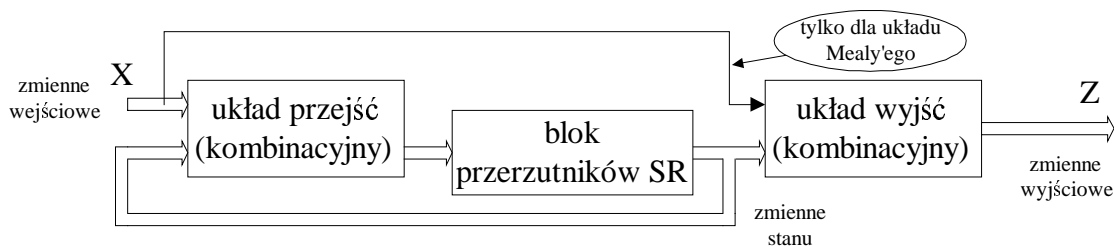


Rys. 14.10 Ogólny model układu asynchronicznego

Ogólny model (strukturę blokową) układu asynchronicznego przedstawiliśmy na rys. 14.10. W odróżnieniu od struktury z rys. 14.4, w modelu tutaj omawianym wyodrębniono dodatkowy układ wyjść, który generuje wartości zmiennych wyjściowych Z.

Jeżeli zmienne wyjściowe zależą od zmiennych wejściowych X i zmiennych stanu Y ($Z=f(X;Y)$), układ będziemy nazywać układem typu Mealy'ego [1], a jeżeli zależą jedynie od zmiennych stanu ($Z=f(Y)$) - układem typu Moore'a [2].

Model przedstawiony na rys.14.10 jest ogólny. Na sprzężenia zwrotne nie narzuca się żadnych ograniczeń oprócz tego, by liczba linii sprzężenia zwrotnego była równa liczbie zmiennych stanu. W przypadku gdy wartości zmiennych stanu pamiętane są w przerzutnikach, można zaproponować model układu pokazany na rys. 14.11.



Rys. 14.11 Ogólny model układu asynchronicznego z przerzutnikami asynchronicznymi

14.5 Układ wyjść

W poprzednich punktach skupiliśmy się na analizie układu asynchronicznego, uwzględniając jedynie zmienne stanu. W podrozdziale niniejszym zajmiemy się układem wyjść. Jak już wspomnieliśmy, wartość zmiennych wyjściowych Z zależy od zmiennych stanu i zmiennych wejściowych (układ Mealy'ego) albo tylko od zmiennych stanu (układ Moore'a).

a)		x_1x_2				
y_1y_2		00	01	11	10	
	00	①/00	①/00	①/01	3	
	01	②/01	1	—	3	
	11	1	4	③/10	③/10	
	10	2	④/01	3	—	

b)		x_1x_2				
		00	01	11	10	Z
	1	①	①	4	3	00
	2	②	1	—	3	01
	3	1	5	③	③	10
	4	—	1	④	3	01
	5	2	⑤	3	3	01

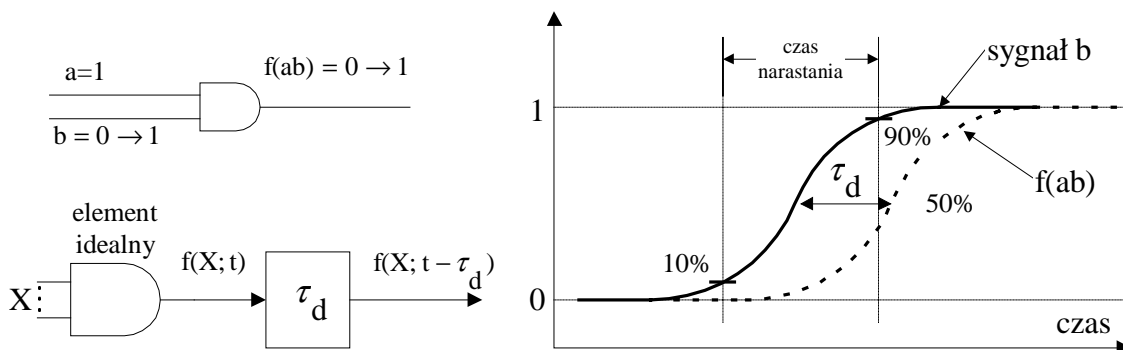
Rys. 14.12 Tablice Huffmana z zaznaczonymi wartościami wyjść: a - dla układu Mealy'ego, b - dla układu Moore'a

Do opisu wyjść układu wykorzystamy tablicę Huffmana. Jest oczywiste, że wartości zmiennych wyjściowych układu typu Mealy'ego możemy wpisać wprost do tablicy Huffmana obok stanów stabilnych (i niestabilnych), ponieważ tablica to nic innego jak zależność $Y(X;y)$, a funkcja wyjść to zależność $Z(X;y)$. Dla układu typu Moore'a funkcja wyjść ma postać $Z(y)$, dlatego nie możemy umieścić jej wprost w tablicy Huffmana. W związku z tym dopisujemy tyle kolumn (zazwyczaj z prawej strony tablicy), ile jest zmiennych wyjściowych, ponieważ wartości wyjść wpisane do tych kolumn będą zależeć tylko od wartości zmiennych stanu. Przykładowe tablice Huffmana z naniesionymi wartościami zmiennych wyjściowych pokazaliśmy na rys. 14.12.

14.6 Hazard i wyścigi

Przed przystąpieniem do wyjaśnienia zjawiska hazardu zwróćmy uwagę na istotę opóźnień wprowadzanych przez elementy logiczne i linie je łączące. W dotychczasowych rozważaniach zazwyczaj przyjmowaliśmy, że wartość sygnału zmienia się natychmiast (jako konsekwencja braku parametru czasu w algebrze Boole'a). Wiemy jednak, że w rzeczywistych warunkach nie możemy uzyskać tego rodzaju zmian. Przyjmiemy zatem, że wymuszenie zmiany stanu elementu logicznego trwa pewien czas.

Odcinek czasu τ_d (rys. 14.13b) odpowiada czasowi opóźnienia wprowadzonego przez rozważany element. Jest to czas, który upłynie od chwili spełnienia warunków pobudzenia elementu przez sygnał wejściowy (tzw. wartość progowa) do chwili otrzymania reakcji na pobudzenie.



Rys. 14.13 Czas opóźnienia elementu logicznego

Z tego względu czas ten nazywamy opóźnieniem tranzytowym, w odróżnieniu od opóźnień świadomie wprowadzonych (elementy opóźniające).

Należy także pamiętać, że opóźnieniami tranzytowymi charakteryzują się linie łączące elementy logiczne, rozumiane jako czasy rozchodzenia się sygnału wzdłuż linii. Dlatego każdy element logiczny (i linie łączące) możemy modelować jako element idealny (reagujący natychmiast) i element opóźniający (rys. 14.13c).

Dokładne wartości opóźnień tranzytowych zazwyczaj nie są znane, co wynika z niejednorodności wykonania elementu logicznego. Przedziały, w których opóźnienia te mogą się znajdować, są zazwyczaj ściśle określone i podawane w katalogach firm produkujących elementy.

Mówimy, że w układzie może wystąpić hazard, jeżeli istnieje możliwa kombinacja wartości opóźnień tranzytowych, która spowoduje pojawienie się nieprawidłowego impulsu wyjściowego. Istotę tego zjawiska wyjaśnimy na przykładzie układu kombinacyjnego przedstawionego na rys. 14.14. Przyjmiemy, że czas opóźnień wprowadzany przez i -ty element układu wynosi τ_i . Załóżmy dalej, że przy $x_1=x_2=x_3=1$ zaczniemy zmieniać wartość $x_2 = 1 \rightarrow 0$. Wartości sygnałów logicznych w poszczególnych punktach układu będą się zmieniać tak, jak to przedstawiliśmy na rys. 14.14c (Dlaczego?).

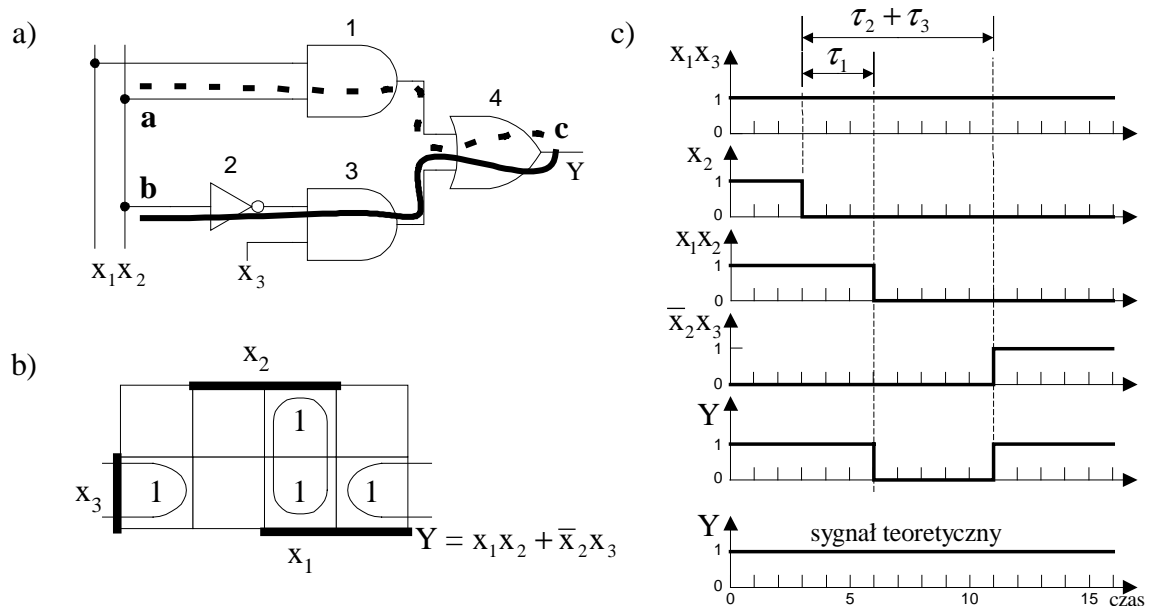
Zauważmy, że na wyjściu układu pojawił się krótkotrwały impuls, będący wynikiem różnych opóźnień sygnału na drogach a-c i b-c. W rozważanym przypadku różnica ta wynika być może z opóźnienia wprowadzonego przez bramkę NOT na drodze b-c. Zjawisko to nazywamy hazardem statycznym, ponieważ sygnał wyjściowy zmienił się parzystą liczbę razy ($1 \rightarrow 0 \rightarrow 1$).

Podkreślimy w tym miejscu, że hazard statyczny może (ale nie musi) pojawić się w układzie, który na różnych drogach (od wejścia do wyjścia) ma jednakową liczbę bramek tego

samemu rodzaju. W takim przypadku wystąpienie zjawiska hazardu oznacza, że różnice w opóźnieniach wynikają z rozrzutu tolerancji czasów opóźnień wyprodukowanych elementów. Należy zatem pamiętać, że hazard to jedynie możliwość wystąpienia błędnej pracy układu (zachowania niezgodnego ze specyfikacją). Z tego też powodu w niektórych podręcznikach spotyka się nazwę ryzyko.

Dla celów dalszych rozważań wygodnie jest powiązać stan stabilny ze stanem wejść (zestaw wartości przyjmowanych przez zmienne wejściowe). Para (stan, stan wejść) jednoznacznie identyfikuje kratkę w tablicy Huffmana i jest krótko nazywana stanem całkowitym. Z tego względu w tablicach Huffmana będziemy różnie oznaczać stany stabilne leżące w tym samym wierszu.

Jeżeli w rozważanym układzie kombinacyjnym z rys. 14.14 połączymy wejście x_3 z wyjściem, otrzymamy układ asynchroniczny z odpowiadającą mu tablicą Huffmana, przedstawioną na rys. 14.15c.

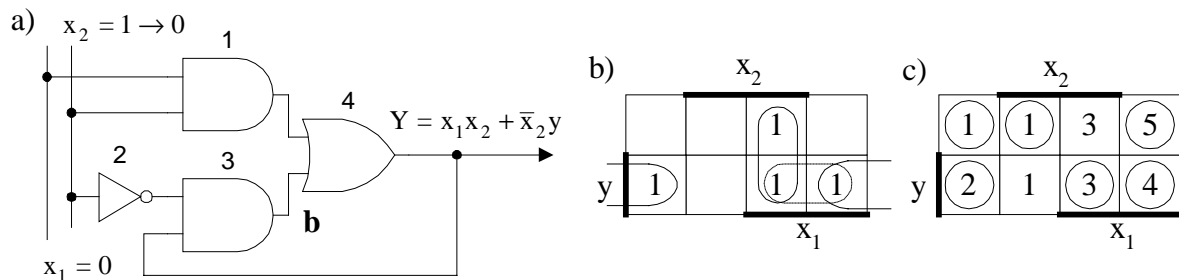


Rys. 14.14 Układ z hazardem statycznym: a - drogi propagacji zmiany sygnału x_2 , b - tablica Karnaugh, c - diagram czasowy

Rozpocznijmy analizę tak otrzymanego układu asynchronicznego zakładając, że znajduje się on w stanie (3). Zmiana wartości zmiennej $x_2 = 1 \rightarrow 0$ powinna (zgodnie z rys. 14.15b) wymusić przejście układu do stanu (4). Załóżmy następnie, że bramki AND i OR reagują znacznie szybciej na zmianę wartości sygnałów wejściowych aniżeli bramka NOT. Założenie to oznacza, że bramka NOT nie zareagowała jeszcze na zmianę wartości x_2 , a zmiana ta spowodowała już zmianę wartości zmiennej $Y = 1 \rightarrow 0$. Zmiana wartości zmiennej Y propaguje się poprzez sprzężenie zwrotne na wejście bramki nr 3. Nietrudno zauważyć, że bez względu na to, jaką wartość przyjmie teraz sygnał wyjściowy bramki NOT, sygnał wyjściowy bramki nr 3 nie ulegnie zmianie. Ponieważ nie istnieje już inne źródło wymuszenia zmian wartości sygnałów w układzie, a $Y = y$, więc układ osiągnął stan stabilny. Stanem tym jest $x_1x_2y=100$, co odpowiada stanowi stabilnemu oznaczonemu jako (5). Jest to stan różny od stanu, do którego układ miał przejść.

Przykład ten ilustruje niepożądany wpływ hazardu statycznego na działanie układu, dlatego należy go eliminować. W omawianym układzie wystarczy tak dobrać czas opóźnień bramek, aby był spełniony warunek $\tau_1 > \tau_2 + \tau_3$.

Oczywiście, z punktu widzenia realizacyjnego rozwiązanie to wymaga od konstruktora układu jednostkowego doboru elementów logicznych ze względu na wprowadzany przez nie czas opóźnień, bądź wprowadzenie specjalnych elementów opóźniających w wybranych ścieżkach układu. Z praktycznego punktu widzenia nie są to rozwiązania zadowalające. Inne rozwiązanie polega na wprowadzaniu do układu elementów nadmiarowych, które oczywiście nie zmieniają realizowanej funkcji logicznej układu, lecz eliminują zjawisko hazardu. W tym rozwiązaniu projektant nie musi brać pod uwagę opóźnień elementów użytych do realizacji układowej. Podaną propozycję eliminacji zjawiska hazardu statycznego ilustruje rys. 14.16. Przykład dotyczy eliminacji hazardu w omawianym wyżej układzie z rys. 14.15.



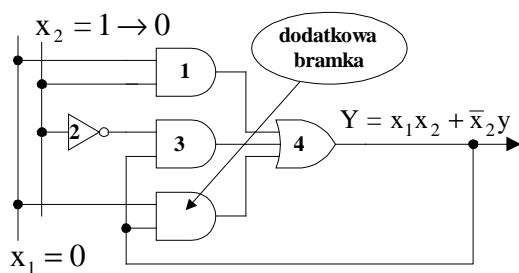
Rys. 14.15 Układ asynchroniczny z hazardem statycznym: a - przykład układu, b - tablica Karnaugh dla funkcji $Y(x_1x_2y)$, c - tablica Huffmana

Dodatkowo wprowadzony element realizujący iloczyn x_1y , jest nadmiarowy, ponieważ

$$Y(x_1x_2y) = x_1x_2 + \bar{x}_2y + x_1y = x_1x_2 + \bar{x}_2y + x_1\bar{x}_2y + x_1x_2y = x_1x_2(1+y) + \bar{x}_2y(1+x_1) = x_1x_2 + \bar{x}_2y$$

czyli otrzymaliśmy pierwotną postać funkcji.

Zauważmy także, że iloczyn x_1y opisuje grupę jedynek wspólną dla obu przylegających do siebie grup opisujących iloczyny x_1x_2 i \bar{x}_2y (przerwana obwódka z rys. 14.15b). Jako



Rys. 14.16 Eliminacja hazardu statycznego

ćwiczenie proponuje się Czytelnikowi przeanalizowanie zachowania się układu na diagramie czasowym.

Eliminacja hazardu statycznego jest jak widać, związana z deminimalizacją wyrażenia boolowskiego. Na rys. 14.17 przedstawiliśmy ogólny algorytm eliminacji hazardu ze względu na zmienną x_i , którego zaletą jest to, że nie wymaga stosowania tablicy Karnaugh.

Przykład:

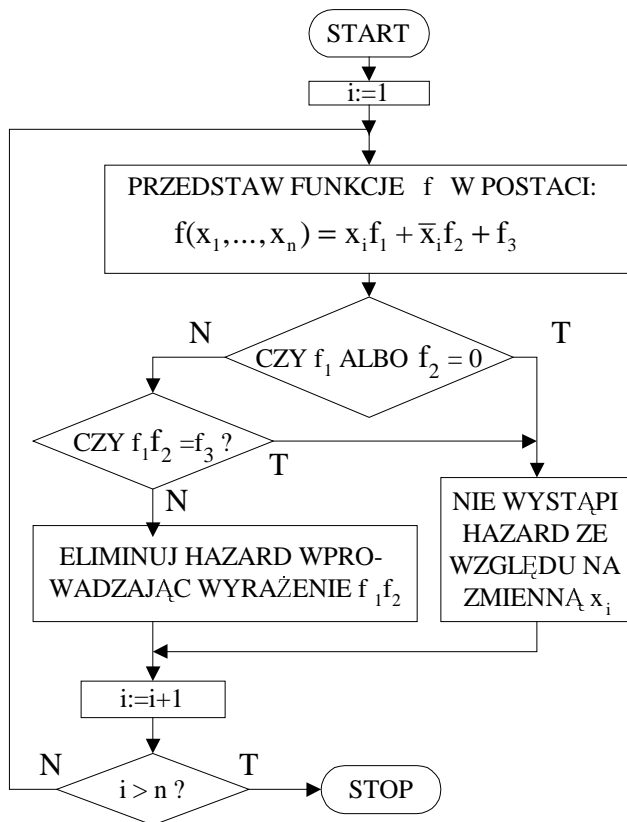
Zastosujemy algorytm poszukiwania możliwości wystąpienia hazardu na przykładzie z rys. 14.14, czyli dla funkcji $f(x_1x_2x_3) = x_1x_2 + \bar{x}_2x_3$.

Krok 1. $i = 1$; $f(x_1x_2x_3) = x_1(x_2) + (\bar{x}_2x_3)$, czyli $f_1 = x_2$, $f_2 = 0$, $f_3 = \bar{x}_2x_3$. Ponieważ $f_2 = 0$, więc nie występuje hazard ze względu na x_1 .

Krok 2. $i = 2$; $f(x_1x_2x_3) = x_2(x_1) + \bar{x}_2(x_3)$, czyli $f_1 = x_1$, $f_2 = x_3$, $f_3 = 0$. Ponieważ f_1 i $f_2 \neq 0$ oraz $f_1 \cdot f_2 \neq f_3$, może wystąpić hazard ze względu na x_2 , który eliminujemy za pomocą funkcji $f_1 \cdot f_2 = x_1x_3$.

Krok 3. $i = 3$; $f(x_1x_2x_3) = x_3(\bar{x}_2) + x_1x_2$, czyli $f_1 = \bar{x}_2$, $f_2 = 0$, $f_3 = x_1x_2$. Ponieważ $f_2 = 0$, więc nie występuje hazard ze względu na x_3 .

Hazardem dynamicznym nazywamy zjawisko polegające na kolejnym wygenerowaniu nieparzystej liczby zmian (>1) wartości sygnału wyjściowego w przypadku, gdy wymagana była pojedyncza jego zmiana.



Rys. 14.17 Algorytm poszukiwania hazardu

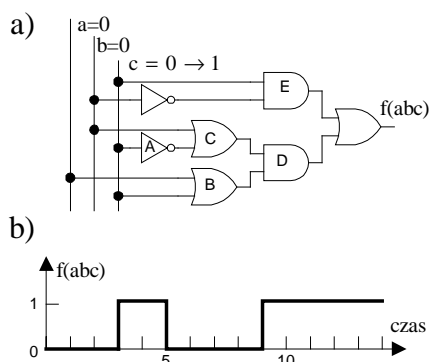
Hazard statyczny powstał jako konsekwencja istnienia dwóch dróg z różnymi opóźnieniami tranzytowymi, po których pobudzenie wywołane jednostkową zmianą wartości sygnału wejściowego propaguje się do rozważanego wyjścia.

Hazard dynamiczny jest rozszerzeniem pojęcia hazardu statycznego dla przypadku istnienia co najmniej trzech takich dróg. W idealnym układzie, w którym pojedyncza zmiana wejścia powinna wymusić jednokrotną zmianę wartości sygnału wyjściowego (na przykład $0 \rightarrow 1$), w układzie rzeczywistym wymusi trzykrotną jego zmianę (na przykład $0 \rightarrow 1 \rightarrow 0 \rightarrow 1$). Spróbuj tak dobrać opóźnienia bramek A, B, C oraz D z rys. 14.18, aby taką trzykrotną zmianę wymusić.

Eliminacja hazardu dynamicznego jest o wiele bardziej skomplikowana i z tego powodu proponuje się projektowanie dwupoziomowych układów logicznych, w których eliminacja hazardu statycznego eliminuje hazard dynamiczny.

Twierdzenie 14.6.1

Dwupoziomowa realizacja układowa AND-OR funkcji przełączającej jest wolna od zjawiska hazardu statycznego, jeżeli zawiera realizacje wszystkich implikantów prostych danej funkcji.



Rys. 14.18 Ilustracja hazardu dynamicznego: a - układ, b - diagram czasowy

Innymi słowy, jeżeli będziemy realizować zadaną funkcję przełączającą, wykorzystując do tego celu wszystkie implikanty proste, będziemy mieli pewność, że nie wystąpi zjawisko hazardu statycznego.

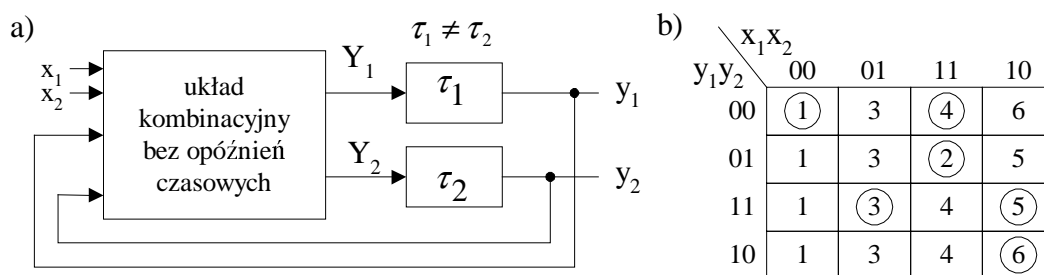
UWAGA: Zjawisko hazardu omawiane jest w rozdziale o układach asynchronicznych, ponieważ chcieliśmy pokazać przede wszystkim niekorzystny wpływ hazardu na działanie tych układów. Podkreślmy jednak, że zjawisko to jest charakterystyczne dla układów kombinacyjnych i moglibyśmy je omówić wcześniej.

Na rys. 14.19 pokazaliśmy blokową strukturę układu asynchronicznego, w której wyodrębniliśmy czasy opóźnień wnoszonych przez układy kombinacyjne obejmowane pętlami sprzężeń zwrotnych. Dla uproszczenia rozważymy tylko dwie linie sprzężenia zwrotnego oraz założymy, że układ zachowuje się zgodnie z tablicą zaproponowaną na rys 14.19b.

Przyjmijmy, że układ znajduje się w stanie (3) ($x_1x_2y_1y_2=0111$) oraz następuje zmiana stanu wejść $x_1x_2=01$ na $x_1x_2=11$. Zmiana wartości zmiennych wejściowych powinna przeprowadzić układ do stanu stabilnego (4). W omawianym przypadku stan sprzężeń zwrotnych y_1y_2 powinien zmienić się z 11 na 00.

Wskutek niejednakowych opóźnień τ_1 i τ_2 (zawsze $\tau_1 < \tau_2$ albo $\tau_1 > \tau_2$) zmiana ta nie może być jednoczesna. Wynika z tego, że dla przypadku $\tau_1 < \tau_2$ szybciej ulegnie zmianie wartość y_1 , co oznacza, że przez chwilę uzyskamy stan $x_1x_2y_1y_2=1101$. Jest to jednak stan stabilny (2), czyli zamiast do postulowanego stanu (4) układ przeszedł do stanu (2). Gdy $\tau_1 > \tau_2$, układ przejdzie do założonego stanu stabilnego (4) (Dlaczego?).

Pojawia się znowu zależność poprawności działania całego układu od opóźnień wprowadzanych przez układ kombinacyjny. Eliminacją tego zjawiska, nazywanego wyścigiem krytycznym (*critical race*), zajmiemy się w dalszych rozdziałach.



Rys. 14.19 Ilustracja zjawiska wyścigów: a - przykładowy układ, b - przykładowa tablica wtórna Huffmana

Gdy układ znajduje się w stanie (4) i nastąpi zmiana stanu wejść $x_1x_2 = 11 \rightarrow 01$, występuje zjawisko wyścigu niekrytycznego. Przy $\tau_1 > \tau_2$ zostanie wymuszony chwilowy stan $y_1y_2=01$ (drugi wiersz tablicy). Wymusiliśmy przejście do stanu $x_1x_2y_1y_2=0111$, co odpowiada stanowi stabilnemu (3). Jeśli $\tau_1 < \tau_2$, przejdziemy najpierw do wiersza czwartego, a następnie do trzeciego, czyli w rezultacie układ osiągnie stan stabilny (3). Mimo, że sygnały na liniach sprzężenia zwrotnego przyjmowały wartości wyróżnione w różnej kolejności, to jednak efekt końcowy jest ten sam, tzn. osiągnęliśmy wymagany stan (3) - stąd nazwa wyścigu.

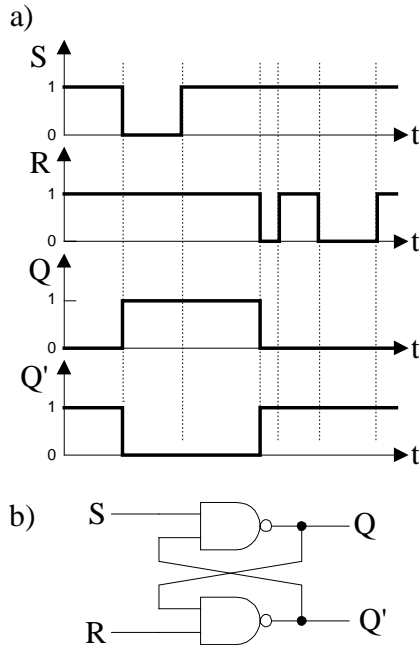
Literatura

- [1] Mealy G.H., A Method for Synthesizing Sequential Circuits, Bell. Sys. Tech. J., vol. 34, Sept. 1955, str. 1045-79
- [2] Moore E.F., Gedanken - Experiments on Sequential Machines, Automata Studies, Annals of Mathematical Studies, no. 34, Princeton University Press, 1956, str. 129-153

ĆWICZENIA

1. Podaj diagram czasowy dla układu z rys. 14.5 uwzględniając opóźnienie Δt . Wskaż stany stabilne i niestabilne.
2. Czy układ z rys. 14.5 można opisać wyrażeniami: $Y^t(x_1^t, x_2^t, y^t) = x_1^t y^t + x_1^t \bar{x}_2^t + x_2^t y^t$, oraz $y^{t+\Delta t} = Y^t$. Co może oznaczać literka t .
3. Czy wykres czasowy przedstawiony na rys. 14.20a odpowiada układowi z rys. 14.20b?
4. Podaj wykresy czasowe i tablice Huffmana dla układów z rys. 14.21.
5. Przeanalizuj zachowanie się układu z rys. 14.22a przy założeniu $\tau_i < \tau_p$ oraz $\tau_i > \tau_p$, gdzie τ_i czas opóźnienia bramki NOT, a τ_p - czas opóźnienia przerzutnika SR. Podaj tablicę Huffmana.

6. Pokaż, że dowolna realizacja funkcji wykorzystująca wszystkie implikanty proste jest bezhazardowa.
7. Podaj własny przykład układu kombinacyjnego, w którym występuje hazard dynamiczny. Wskaż sposób jego wyeliminowania.
8. Przeanalizuj zachowanie się układu z rys. 14.22b przy zmianie $abc=000 \rightarrow 011$. Jeżeli zauważyłeś niekorzystne zjawisko, spróbuj je wyeliminować.



Rys. 14.20 Wykres czasowy układu asynchronicznego

9. Przeanalizuj zachowanie się układu z rys. 14.22c, gdy $a = 0 \rightarrow 1 \rightarrow 0 \rightarrow 1 \rightarrow \dots$

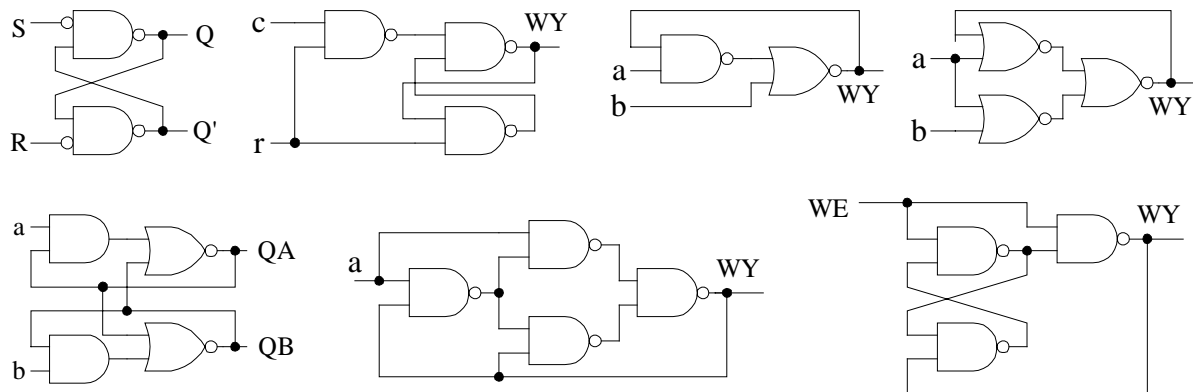
10. Czy w dwustanowym układzie asynchronicznym może wystąpić wyścig?

11. Czy wyścig krytyczny może wystąpić w układzie asynchronicznym, którego tablica wtórna ma w każdej kolumnie tylko jeden stan stabilny?

12. Podaj własny przykład tablicy wtórnej, w której występują wszystkie rodzaje wyścigów.

13. W układzie z rys. 14.23a występuje zjawisko hazardu dynamicznego. Obok każdego elementu logicznego i opóźniającego podaliśmy wielkość opóźnienia wprowadzanego przez dany element (w jednostkach względnych). Należy dobrać wartość opóźnienia odpowiedniego elementu logicznego bądź opóźniającego tak, aby:

- a) zwiększyć czas trwania impulsu A (rys. 14.23b),
 - b) odsunąć tylne zbocze impulsu A od przedniego zbocza impulsu B,
 - c) wyeliminować zjawisko hazardu dynamicznego.
14. Przeanalizuj „ciekawý” układ asynchroniczny o



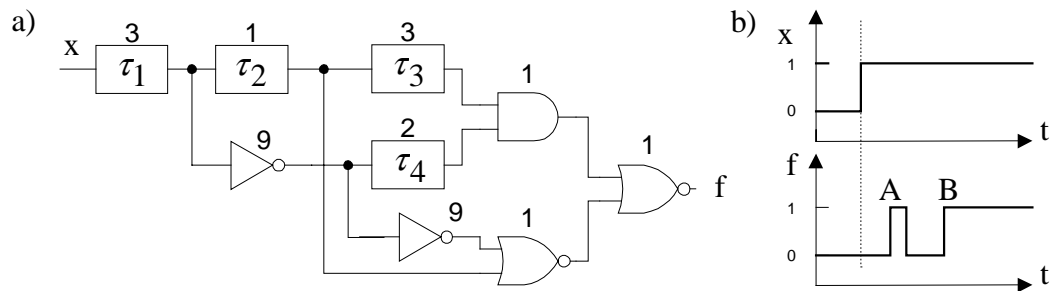
Rys. 14.21 Przykładowe układy asynchroniczne

wejściach D i ZEGAR oraz wyjściu Q, przedstawiony na rys. 14.25. Na początku odszukaj wszystkie pętle sprzężenia zwrotnego (jest ich 3), rozetnij je, napisz wyrażenia logiczne oraz utwórz tablice Huffmana. Czy zachowanie układu jest zgodne z następującym opisem: „wartość sygnału D przenosi się na wyjście Q tylko wtedy, gdy sygnał ZEGAR zmienia swą wartość z 0 na 1”?

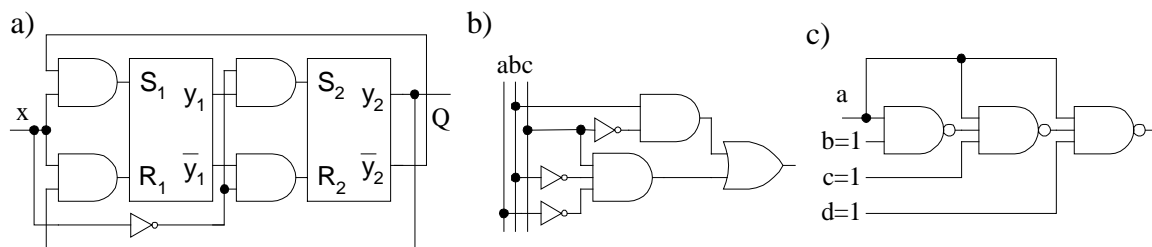
Jak zachowuje się układ, gdy sygnał ZEGAR zmienia swą wartość z 1 na 0? Jaka zależność występuje między sygnałem Q i Q'?

15. Przypomnijmy sobie elementarny układ pamiętający złożony z dwóch bramek NOR. Załóżmy, że każda z bramek wnosi opóźnienie równe t oraz $S=R=0$ ($Q=0$). Narysuj przebieg

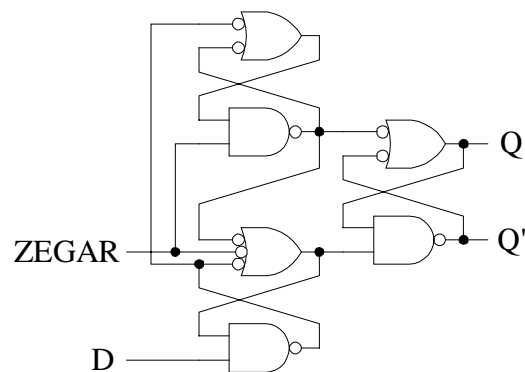
czasowy uwzględniający zmianę $S = 0 \rightarrow 1 \rightarrow 0$ (ustawiamy $Q = 0 \rightarrow 1$), po której następuje zmiana $R = 0 \rightarrow 1 \rightarrow 0$ (gasimy $Q = 1 \rightarrow 0$). Na podstawie przebiegu czasowego ustal minimalny czas trwania $S=1$ by układ został ustawiony, oraz czas trwania $R=1$ by został zgaszony.



Rys. 14.22 a - układ z hazardem dynamicznym, b - diagram czasowy



Rys. 14.23 Przykładowe układy kombinacyjne



Rys. 14.24 "Ciekawy" układ synchroniczny

15 SYNTEZA UKŁADÓW ASYNCHRONICZNYCH

Dotychczas analizowaliśmy zachowanie zadanych układów asynchronicznych, zwracając uwagę na występujące w nich charakterystyczne zjawiska. Właściwie obojętne nam było, czy punktem wyjścia do analizy była wtórna tablica Huffmana, czy realizacja układu. Wynikało to z tego, że mając realizację układu potrafiliśmy otrzymać odpowiadającą mu wtórną tablicę Huffmana.

Celem syntezy układów asynchronicznych jest uzyskanie schematu logicznego układu, czyli utworzenie tablicy Huffmana, pod warunkiem, że opisuje ona zachowanie układu zgodnie ze specyfikacją. Specyfikacja może być zadana słownie lub za pomocą diagramów czasowych przebiegów sygnałów wejściowych i wyjściowych.

15.1 Metoda Huffmana

Przed przystąpieniem do omówienia poszczególnych kroków metody poczynimy założenia, które obowiązywać będą przy syntezie. Należy także pamiętać o nich wykorzystując syntezy układ asynchroniczny w konkretnym zastosowaniu. Otóż założymy, że:

1. w danej chwili zmianie ulega tylko jeden sygnał wejściowy,
2. kolejnej zmiany sygnału wejściowego dokonujemy po osiągnięciu przez układ stanu stabilnego,
3. sygnały mają charakter potencjałowy (nieimpulsowy),

Dwa pierwsze założenia określają pewien rodzaj pracy układu. W literaturze nazywa się on rodzajem podstawowym (*fundamental mode*). Trzecie założenie poczyniliśmy tylko po to, by odróżnić syntezy układ asynchroniczny od układów impulsowych, w których pobudzenie linii wejściowych manifestuje się pojawieniem krótkotrwałego impulsu.

Syntezy układu asynchronicznego będziemy realizować metodą zaproponowaną przez Huffmana. Kolejne kroki metody można przedstawić następująco:

Krok 1. Specyfikacja problemu.

Krok 2. Skonstruowanie pierwotnej tablicy Huffmana.

Krok 3. Minimalizacja liczby stanów (otrzymanie wtórnej tablicy Huffmana).

Krok 4. Zakodowanie stanów z uwzględnieniem zjawiska hazardu i wyścigu.

Krok 5. Utworzenie tablicy przejść i tablicy wyjść.

Krok 6. Napisanie wyrażeń boolowskich opisujących układ.

Krok 7. Narysowanie układu.

15.1.1 Specyfikacja problemu

Specyfikacja problemu jest istotna przy wszelkiego rodzaju systemach, zwłaszcza złożonych, gdzie należy precyzyjnie określić zachowanie się syntezywanego systemu, nie pozostawiając miejsca na dowolną jego interpretację. Najbardziej rozpowszechnionymi specyfikacjami zachowania się układów asynchronicznych są: opis słowny, wykres czasowy oraz tablica wtórna. Wtórna tablica Huffmana jest formalną postacią specyfikacji, nie dającą możliwości interpretacyjnych. Wykres czasowy jest także precyzyjną specyfikacją zachowania układu, chętnie wykorzystywaną w firmowych katalogach układów logicznych. Dla naszych celów wykorzystamy opis, uwzględniając jego walor dydaktyczny polegający na tym, że specyfikacja w postaci opisu słownego może okazać się nieprecyzyjna, dopuszczając różną interpretację założeń przy niestarannym sformułowaniu zagadnienia.

Kolejne kroki metody Huffmana pokażemy syntezyując układ, który powinien zachowywać się zgodnie z następującym opisem słownym: wartość sygnału wyjściowego $z=0$, gdy $x_2=0$;

$z=1$, gdy zmienie ulegnie sygnał x_1 (jeżeli $x_2=1$) i pozostanie 1 aż do chwili, gdy x_2 przyjmie wartość 0. Z opisu słownego wynika, że układ asynchroniczny, którego realizacją układową mamy podać, ma dwie zmienne wejściowe x_1 i x_2 oraz jedną zmienną wyjściową z .

15.1.2 Pierwotna tablica Huffmana

Pierwotną tablicę Huffmana będziemy nazywać tablicę, której wiersze odpowiadają pojedynczym stanom stabilnym, a kolumny poszczególnym zestawom zmiennych wejściowych (stanom wejść). Należy w taki sposób wpisać do tablicy stany stabilne i niestabilne, aby tablica odzwierciedlała zachowanie układu zgodnie z opisem słownym (z doświadczenia wynika, że jest to najtrudniejszy etap syntezy układu asynchronicznego).

Przyjmijmy zatem, że na przecięciu pierwszego wiersza i pierwszej kolumny będzie wpisany stan stabilny (1), od którego rozpoczniemy wypełnianie tablicy (stan początkowy). Stan ten oznacza, że $x_1x_2=00$ oraz $z=0$, ponieważ - zgodnie z opisem słownym - $z=0$ zawsze, gdy $x_2=0$ (wartość sygnału z zapisujemy w tablicy po kresce ukośnej). Niech $x_1 = 0 \rightarrow 1$. Przesuwamy się do czwartej kolumny $x_1x_2=10$, wpisujemy na przecięciu z pierwszym wierszem stan niestabilny 2, a w wierszu drugim tej samej kolumny stan stabilny (2). Nastąpiła zmiana wartości zmiennej x_1 , ale nadal $x_2=0$, czyli nie zmieniamy wartości sygnału z . Dopuszczamy teraz do zmiany $x_2 = 0 \rightarrow 1$. Przesuwamy się do trzeciej kolumny $x_1x_2=11$, wpisujemy stan niestabilny 3, a w wierszu trzecim stan stabilny (3). Chociaż $x_2=1$, lecz nie nastąpiła zmiana wartości sygnału x_1 , czyli nadal $z=0$. Zmieńmy zatem wartość zmiennej $x_1 = 1 \rightarrow 0$, wymuszając przejście do stanu stabilnego (4). Oczywiście $z=1$, ponieważ nastąpiła zmiana wartości zmiennej x_1 oraz $x_2=1$. Jeżeli teraz $x_2 = 1 \rightarrow 0$, to $z=0$ i przejdziemy do stanu (5). Wypełnioną tablicę przedstawiono na rys. 15.1a (górna część tablicy, nad linią przerywaną). Powróćmy do stanu stabilnego (1) i założmy, że $x_2 = 0 \rightarrow 1$ (poprzednio $x_2 = 1 \rightarrow 0$). Przejdziemy do stanu (6). Zmienna $x_2=1$, ale nie ma zmiany wartości sygnału x_1 , więc $z=0$. Niech teraz $x_1 = 0 \rightarrow 1$. Przechodzimy do stanu (7) oraz $z=1$ (nastąpiła zmiana sygnału x_1 oraz $x_2=1$). Przy zmianie $x_2 = 1 \rightarrow 0$ przechodzimy do stanu (8) oraz $z=0$.

a)	x_1x_2	00	01	11	10
	(1)/0				2
				3	(2)/0
		4	(3)/0		
	5	(4)/1			
	(5)/0				

b)	x_1x_2	00	01	11	10
	(1)/0	6	—		2
		—	3	(2)/0	
	—	4	(3)/0		
	5	(4)/1		—	
	(5)/0	6	—		
		(6)/0	7	—	
	—		(7)/1	8	
	9	—		(8)/0	
	(9)/0		—		

c)	x_1x_2	00	01	11	10
	(1)/0	6	—		2
	1	—	3	(2)/0	
	—	4	(3)/0		2
	5	(4)/1	7	—	
	(5)/0	6	—		2
	1	(6)/0	7	—	
	—	4	(7)/1	8	
	9	—	3	(8)/0	
	(9)/0	6	—		2

Rys. 15.1 Kolejne etapy wypełniania pierwotnej tablicy Huffmana

Łatwo zauważyć, że dwie osoby tworzące (niezależnie od siebie) tablicę pierwotną mogą otrzymać dwie różne postacie tej tablicy - nie musi to jednak wcale oznaczać, że jedna z tych tablic jest błędna. Obie tablice mogą być poprawne, różniąc się jedynie permutacją wierszy (bądź kolumn) czy oznaczeniami stanów (w wyniku rozpatrywania innych sekwencji wartości

zmiennych w czasie).

Aby tablica z rysunku 15.1 była kompletna, należy wypełnić puste kratki. W miejscach, w których należy dokonać jednoczesnej zmiany obu wartości sygnałów wejściowych (tzn. $x_1x_2=00 \Leftrightarrow 11$, albo $x_1x_2=10 \Leftrightarrow 01$) wpisujemy kreskę (—), ponieważ wiemy, że takiej zmiany wartości sygnałów wejściowych nigdy nie wymusimy. Dlatego kreska w tablicy będzie oznaczać stan obojętny, wyznaczony przez teoretycznie możliwą zmianę wartości sygnałów wejściowych, która w rzeczywistości nigdy nie wystąpi. W praktyce często występują sytuacje, o których z góry wiemy, że odpowiadające im sekwencje zmian wartości sygnałów na wejściu układu nigdy nie wystąpią (oczywiście zakładamy milcząco, że urządzenia generujące sygnały wejściowe są sprawne). Sytuacje takie także zaznaczamy za pomocą kreski (stany nieokreślone). Należy tylko pamiętać o tym, by kreską oznaczać tylko te sekwencje zmian wartości sygnałów wejściowych, które rzeczywiście nie mogą wystąpić.

W następnych krokach syntezy układu asynchronicznego będziemy wykorzystywać stany nieokreślone i obojętne, dlatego dobrze należy zrozumieć ich istotę. W rozważanym przez nas przykładzie występują jedynie stany obojętne. Opis słowny nie wskazuje, by mogły występować stany nieokreślone (wtedy cała kolumna jest wypełniona kreskami). Uzupełnioną kreskami pierwotną tablicę Huffmana przedstawiliśmy na rys. 15.1b.

W pozostałych wolnych miejscach należy wpisać stany niestabilne wymuszające w układzie przejście do zadanego stanu stabilnego (stany dozwolone).

Założmy, że układ jest w stanie (2) i należy wypełnić puste miejsca na przecięciu z pierwszą kolumną ($x_1x_2=00$). Do wyboru mamy przejście do stanów stabilnych (1), (5) oraz (9) (wszystkie znajdują się w pierwszej kolumnie). Wybieramy przejście do (1) jako niesprzeczne z opisem słownym, wpisując w puste miejsce stan niestabilny 1.

a)

x_1x_2	00	01	11	10
1	①/0	6	—	2
2	1	—	3	②/0
3	—	4	③/0	2
4	5 1	④/1	7	—
5	5 /0	6	—	2
6	1	⑥/0	7	—
7	—	4	⑦/1	8 2
8	9	—	3	⑧/0
9	9 /0	6	—	2

b)

x_1x_2	00	01	11	10	Z
1	①	6	—	2	0
2	1	—	3	②	0
3	—	4	③	2	0
4	1	④	7	—	1
5	1	⑥	7	—	0
6	—	4	⑦	2	1

Rys. 15.2 Poszukiwanie stanów pseudorównoważnych: a - usuwanie stanów, b - tablica Huffmana po usunięciu stanów

Założmy, że układ znajduje się w stanie (7) oraz $x_1=1 \rightarrow 0$. W sytuacji tej mamy do wyboru przejście do stanu stabilnego (4)/1 albo (6)/0. W rozważanym stanie (7) wartość sygnału wyjściowego $z=1$ powinna pozostać taka do czasu, gdy x_2 przyjmie wartość 0. Wartość zmiennej x_2 pozostała bez zmian, należy więc na przecięciu wiersza 7 i kolumny $x_1x_2=01$ wpisać stan niestabilny 4, aby być w zgodzie z opisem słownym. Postępując analogicznie, otrzymamy wypełnioną pierwotną tablicę Huffmana podaną na rys. 15.1c.

Zwróćmy uwagę na to, że tablica pierwotna w każdym wierszu ma tylko jeden stan

stabilny, oraz że wartość wyjścia jest przypisana jedynie do stanów stabilnych (przyporządkowaniem wartości sygnałów wyjściowych stanom niestabilnym zajmiemy się dalej). Wynika z tego, że tablica pierwotna zawsze opisuje układ typu Moore'a.

15.1.3 Minimalizacja liczby stanów

Analizując w poprzednich rozdziałach układy asynchroniczne korzystaliśmy z wtórnej tablicy Huffmana. Każdy wiersz tablicy miał przypisane z lewej strony zestawy wartości zmiennych stanu $y_1y_2...y_n$. W odróżnieniu od pierwotnej tablicy Huffmana, w wierszu tablicy wtórnej może występować kilka stanów stabilnych. Zmienna stanu reprezentowana jest przez pojedynczą pętlę sprzężenia zwrotnego, każda zaś pętla to układ kombinacyjny nią obejmowany. Jeżeli zatem potrafimy zmniejszyć liczbę wierszy tablicy pierwotnej, nie zmieniając oczywiście zachowania się układu przez nią opisywanego, to zmniejszymy koszt układu eliminując tę jego część, która musiałaby wzbudzać wyeliminowane sprzężenie zwrotne.

Proces redukcji liczby wierszy tablicy pierwotnej nazwiemy jej minimalizacją. W rezultacie minimalizacji otrzymamy tablicę wtórną.

Dwa stany (wiersze tablicy pierwotnej) nazywamy pseudorównoważnymi, gdy jednocześnie spełnione są warunki:

1. stany stabilne znajdują się w tych samych kolumnach,
2. wartości zmiennych wyjściowych odpowiadające obu stanom są niesprzeczne,
3. dla rozważanych stanów oraz przy dowolnej zmianie stanu wejść układ przechodzi z rozpatrywanych stanów do stanów niesprzecznych lub pseudorównoważnych.

a)

x_1x_2	00	01	11	10	Z
	4	—	2	①	0
	—	3	②	14	0
	4	③	13	—	0
	④	5	—	1	0
	4	⑤	6	—	0
	—	12	⑥	7	1
	4	—	6	⑦	1
	10	⑧	6	—	1
	10	—	6	⑨	1
	⑩	8	—	9	1
	—	12	⑪	7	1
	10	⑫	11	—	1
	—	3	⑬	1	0
	4	—	13	⑭	0

b)

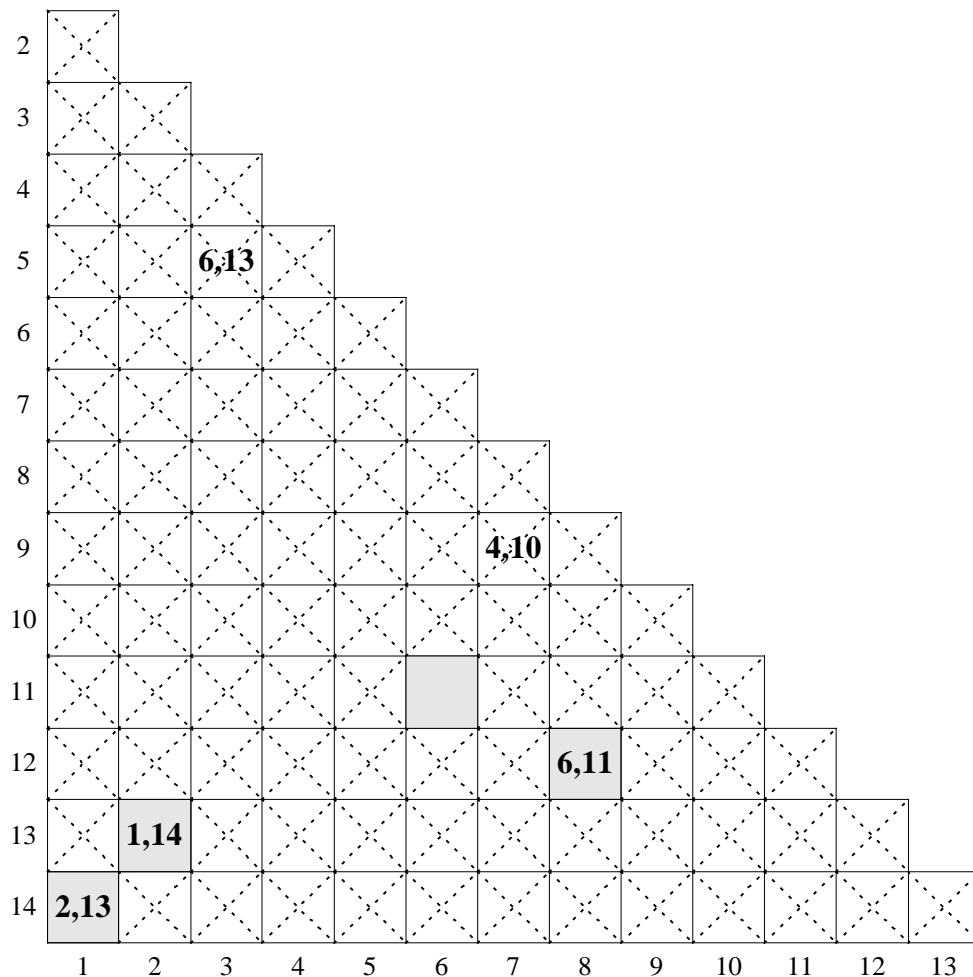
x_1x_2	00	01	11	10	Z
	4	—	2	①	0
	—	3	②	14 ₁	0
	4	③	13 ₂	—	0
	④	5	—	1	0
	4	⑤	6	—	0
	—	12	⑥	7	1
	4	—	6	⑦	1
	10	⑧	6	—	1
	10	—	6	⑨	1
	⑩	8 ₁₂	—	9	1
	—	12	⑪	7	1
	10	⑫	11 ₆	—	1
	—	3	⑬	1	0
	4	—	13	⑭	0

Rys. 15.3 Wykrywanie stanów pseudorównoważnych: a - przykładowa pierwotna tablica Huffmana, b - usuwanie stanów pseudorównoważnych

Dwa stany albo dwa sygnały są niesprzeczne, gdy są one jednakowe lub przynajmniej jeden z nich jest nieokreślony (—). Zwróćmy uwagę, że powyższa definicja stanów pseudorównoważnych jest rekurencyjna.

W pierwszej kolumnie tablicy pierwotnej z rys. 15.1c znajdują się trzy stany stabilne - $(1)/0$, $(5)/0$, $(9)/0$. Rozpatrując je parami - $(1,5)$ $(5,9)$ i $(1,9)$ - stwierdzamy, że warunki 1 i 2 definicji stanów pseudorównoważnych są spełnione. Warunek 3 wymaga, aby w rozważanych parach wierszy były niesprzeczne stany niestabilne lub pseudorównoważne. Zauważamy, że wszystkie stany niestabilne w rozważanych wierszach są jednakowe, a więc stany (1) , (5) i (9) są stanami pseudorównoważnymi (tworzą zbiór stanów pseudorównoważnych). Zbiór stanów pseudorównoważnych możemy zastąpić jednym nowym stanem (Dlaczego?). Załóżmy, że reprezentantem zbioru będzie stan stabilny (1) . Wykreślamy z tablicy wiersz 5 i 9, a w miejscach występowania tych stanów niestabilnych w pozostałej części tablicy wpisujemy stan niestabilny 1 (łatwo zauważyć, że zmieniamy tylko pierwszą kolumnę tablicy). Po dokonaniu tej operacji, zilustrowanej na rys. 15.2a, zauważamy, że także stany (2) i (8) są stanami pseudorównoważnymi. Wykreślamy wiersz 8 i zmieniamy stany 8 na 2. Tę operację ilustruje rys. 15.2b.

Ze względu na niespełnienie warunku 2, pary stanów $(4,6)$ i $(3,7)$ nie są parami stanów pseudorównoważnych. W rezultacie otrzymaliśmy tablicę przedstawioną na rys. 15.2b. Dodatkowo dokonaliśmy przeniesienia wartości zmiennej wyjściowej do osobnej kolumny z.

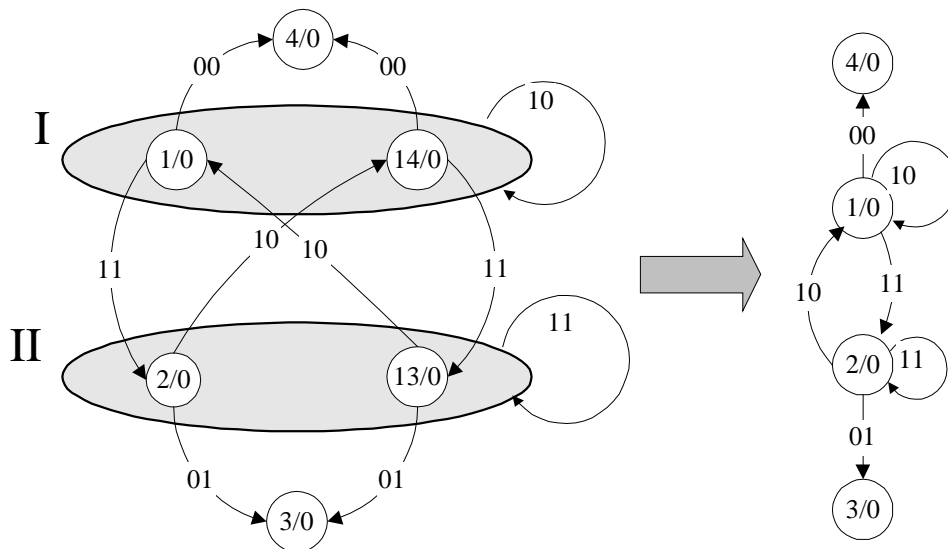


Rys. 15.4 Tablica trójkątna

Weźmy fikcyjny przykład, specjalnie przygotowany w celu pełniejszego wyjaśnienia istoty stanów pseudorównoważnych (patrz rys. 15.3). Stany 4 i 10 spełniają warunek 1, lecz nie spełniają warunku 2, nie są zatem pseudorównoważne. Kontynuując porównywanie wszystkich możliwych par stanów stabilnych znajdujących się w tej samej kolumnie, przeko-

ujemy się, że para (6,11) jest pseudorównoważna. Wykreślamy zatem wiersz 11 oraz tam, gdzie jest 11, wpisujemy 6. W konsekwencji para (8,12) jest pseudorównoważna. Mówimy, że pseudorównoważność pary (8,12) uwarunkowana jest równoważnością pary (6,11).

Pewna trudność występuje, gdy rozważymy parę (1,14). Para stanów jest pseudorównoważna pomimo niezgodności stanów niestabilnych występujących w kolumnie $x_1x_2=11$ (występują tam stany 2 i 13). Badając z kolei pseudorównoważność pary (2,13) widzimy, że stany te są równoważne pod warunkiem pseudorównoważności pary (1,14). Łańcuch uwarunkowań zamknął się. Należy z tego wnosić, że pary (1,14) i (2,13) są pseudorównoważne. Zredukowaną tablicę podaliśmy na rys. 15.3b. Jeszcze większe trudności występują, gdy tablica pierwotna będzie miała duże rozmiary. Celem zmniejszenia możliwości pomyłki możemy uciec się do pomocy tablicy trójkątnej, zaproponowanej przez Ungera i Paulla. Za jej pomocą wykrywamy wszystkie pary stanów pseudorównoważnych zaznaczając spełnienie któregoś z warunków 1, 2 oraz 3 (puste miejsce), niespełnienie któregoś z warunków (wpisujemy X) albo zaznaczając pseudorównoważność uwarunkowaną pseudorównoważnością innej pary stanów (rys. 15.4). Z wypełnionej tablicy trójkątnej wynika, że para (7,9) jest pseudorównoważna pod warunkiem pseudorównoważności pary (4,10). Para (4,10) nie jest jednak pseudorównoważna (nie jest spełniony warunek 2), dlatego na przecięciu kolumny 7 z wierszem 9 wpisujemy znak X (przekreślamy wpisaną tam poprzednio parę (4,10)). Analogiczna sytuacja występuje dla pary (3,5). Pary stanów (6,11), (8,12), (1,14) oraz (2,13) są pseudorównoważne, co zgodne jest z poprzednio otrzymanym wynikiem.



Rys. 15.5 Grafowa ilustracja redukcji stanów pseudorównoważnych

Uważny Czytelnik powinien zauważyć, iż bez większego uzasadnienia przyjęliśmy, że pary stanów pseudorównoważnych łączymy w jeden stan. Spróbujmy zilustrować poprawność takiego założenia. Rozważmy zatem pary stanów pseudorównoważnych (1,14) i (2,13), pomagając sobie grafem przedstawionym na rys. 15.5a, który jest inną reprezentacją fragmentu tablicy pierwotnej z rys. 15.3 (dlaczego fragmentu?). Stany pseudorównoważne (1,14) oraz (2,13) obwiedliśmy, zaznaczając obok wartości sygnałów wejściowych x_1x_2 . Każdemu ze stanów przypisaliśmy wartość wyjścia oraz zaznaczyliśmy nad strzałkami wartości sygnałów wejściowych powodujące przejście do następnego stanu stabilnego. Na rys. 15.5b przedstawiliśmy graf zawierający mniejszą liczbę wierzchołków, który pod względem generowanych wyjść odzwierciedla identyczne zachowanie układu z rys. 15.5a. Przejście między obwódkami $I \Leftrightarrow II$ jest wymuszone przez sekwencję zmian wartości sygnałów

$x_1x_2=11-10-11-10-\dots$ Przy identycznej sekwencji zmian wartości na wejściu będziemy przechodzić między stanami (1)/0 a (2)/0 oraz (2)/0 a (1)/0 (rys. 15.5b).

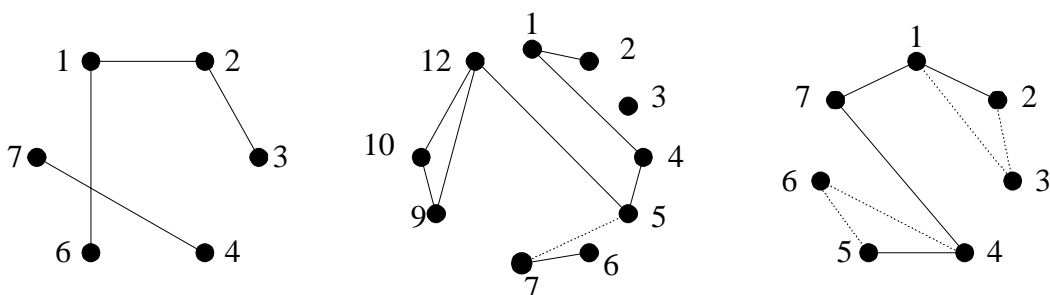
Zauważmy dalej, że będąc w którymkolwiek ze stanów wewnątrz obwódki I i zmieniając $x_1x_2=10$ na $x_1x_2=00$, przejdziemy do stanu (4)/0. Będąc w którymkolwiek ze stanów obwódki II, po zmianie $x_1x_2=11 \rightarrow 01$ przejdziemy do stanu (3)/0. Identyczne przejścia uzyskamy wychodząc odpowiednio ze stanu (1)/0 oraz (2)/0, po uwzględnieniu grafu z rys. 15.5b. Oba grafy dają nam zatem w miejscach identyczną informację o zachowaniu układu asynchronicznego. Łatwo zauważyć, że stany (1,14) oraz (2,3) z rys. 15.5a zastąpiliśmy odpowiednio stanami (1) oraz (2) z rys. 15.5b. Poszukując par stanów pseudorównoważnych, otrzymujemy zredukowaną postać pierwotnej tablicy Huffmana, zawierającą minimalną liczbę stanów stabilnych (liczba wierszy jest jednak nadal równa liczbie stanów stabilnych).

Kolejny proces skracania liczby wierszy tablicy pierwotnej będzie sprowadzać się do takiego połączenia wszystkich par stanów (wierszy tablicy), które mogą być reprezentowane przez ten sam wektor stanu $y_1y_2\dots y_n$, nie powodując zmiany zachowania się układu. Innymi słowy, przy łączeniu dwóch wierszy tablicy w jeden, w każdej kolumnie musi być spełniony jeden z warunków: (S_i, S_i) , $(S_i, -)$, $(-, S_i)$ lub $(-, -)$, gdzie S_i oznacza stan stabilny albo niestabilny. Na rys. 15.6 pokazaliśmy przykładową parę wierszy i wynik ich połączenia.

				Z
5	→	6	1	0
5	→	6	1	1
5	→	6	1	

Rys. 15.6 Skracanie tablicy pierwotnej

Zauważmy, że w przykładzie z rys. 15.6 połączyliśmy wiersze pomimo różnicy występującej na wyjściu (układ typu Mealy'ego). Gdy w procesie łączenia wierszy uwzględnimy wartość sygnałów wyjściowych, otrzymamy układ typu Moore'a. Stany, które możemy połączyć według powyższych zasad nazywamy stanami zgodnymi. Nowy wiersz, uzyskany w rezultacie połączenia dwóch stanów zgodnych, może być użyty do połączenia z innym wierszem. Z tego względu w procesie poszukiwania maksymalnych zbiorów stanów zgodnych pomocny jest wykres skracania (wykres zgodności). Na wykresie tym łączymy linią ciągłą punkty odpowiadające stanom zgodnym i mającym jednakowe wyjścia. Linią przerywaną łączymy dwa punkty odpowiadające parze stanów zgodnych, nie uwzględniając przy tym wartości wyjść. Pierwszy przypadek dotyczy realizacji układu typu Moore'a, a drugi zaś układu typu Mealy'ego.

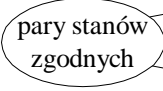


Rys. 15.7 Wykresy skracania

Zazwyczaj na jednym wykresie skracania nanosimy linie ciągłe i przerywane, ponieważ ułatwia to wybór typu układu. Dodajmy jeszcze, że linia ciągła traktowana jest także jako linia przerywana, jeżeli rozpatrujemy układ Mealy'ego, lecz nie na odwrót (Dlaczego?). Na rys. 15.7a przedstawiliśmy wykres skracania dla syntetyzowanego przez nas przykładu (wykresy z rys. 15.7b i c są fikcyjne). Dla wykresu z rys. 15.7a rodzina zbiorów maksymalnych

stanów zgodnych pokrywających wszystkie stany tablicy (w jej skład muszą wchodzić wszystkie stany) jest następująca: układ Moore'a i Mealy'ego - $\{(2,3), (1,6), (4,7)\}$.

Z kolei dla wykresu z rys. 15.7b otrzymamy: układ Moore'a i Mealy'ego - $\{(9,10,12), (6,7), (4,5), (1,2), (3)\}$, a z rys. 15.7c: układ Moore'a - $\{(1,2), (4,7), (3), (5), (6)\}$, $\{(1,7), (4,5), (2), (3), (6)\}$, układ Mealy'ego - $\{(1,2,3), (4,5,6), (7)\}$.



x_1x_2	00	01	11	10	Z
(2,3)	1	4	③	②	0
(1,6)	①	⑥	7	2	0
(4,7)	1	④	⑦	2	1

Rys. 15.8 Tablica wtórna

jących wpisane wszystkie przekątne. Na rys. 15.8 przedstawiliśmy wtórną tablicę Huffmana uwzględniając wykres skracania z rys. 15.7a.

Zaznaczmy w tym miejscu, że sposób „poruszania” się po tablicy wtórnej pozostaje identyczny jak dla tablicy pierwotnej. Należy jednak pamiętać, że nie możemy dokonywać zmian stanu wejść, które w rzeczywistości nie występują albo są zabronione. Musimy o tym pamiętać, ponieważ na etapie łączenia par stanów zgodnych niektóre kreski zostały wchłonięte. Proponuje się Czytelnikowi sprawdzenie zgodności otrzymanej tablicy wtórnej z opisem słownym.

Na koniec przypomnijmy, że pomimo różnych oznaczeń, stany stabilne występujące w wybranym wierszu tablicy są tymi samymi stanami układu.

Możliwe kodowania

y_1y_2	y_1y_2	y_1y_2	x_1x_2	00	01	11	10
00	00	00	↔	1	4	③	②
01	01	11	↔	①	⑥	7	2
10	11	10	↔	1	④	⑦	2
11	10	01	↔	—	—	—	—

a) b) c)

Rys. 15.9 Propozycje zakodowania tablicy wtórnej

15.1.4 Kodowanie stanów

Proces kodowania sygnałów polega na przypisaniu każdemu wierszowi tablicy wtórnej niepowtarzalnego zestawu wartości zmiennych stanu. Dla n -wierszowej tablicy wymaganych jest q zmiennych stanu, $2^{q-1} < n \leq 2^q$. Jeżeli n jest potęgą dwójki, wszystkie możliwe kombinacje zmiennych stanu będą jednoznacznie identyfikować wszystkie wiersze tablicy wtórnej.

Często ogólna liczba możliwych zestawów przewyższa liczbę stanów. Pojawiają się zatem nie wykorzystane zestawy wartości zmiennych stanu. Zachodzi to również w przypadku przez nas rozważanym (rys. 15.8). Jeden zestaw będzie niewykorzystany (Dlaczego?).

Na pierwszy rzut oka proces kodowania wydaje się bardzo prosty. Tak nie jest. Zmienne stanu powinny być w taki sposób przypisane do wierszy tablicy wtórnej, by niemożliwa była „jednoczesna” zmiana więcej niż jednej wartości wybranej zmiennej stanu. Innymi słowy, dowolna zmiana stanu wejść powinna wymusić tylko pojedynczą zmianę sygnału na wybranej pętli sprzężenia zwrotnego. Na rys. 15.9 pokazaliśmy wszystkie (Dlaczego?) możliwe zestawy wartości zmiennych stanu przypisane do rozpatrywanej przez nas tablicy wtórnej.

Sprawdźmy teraz, czy kodowanie (a) jest do przyjęcia, tzn. czy nie wprowadza wyścigu krytycznego. W tym celu należy przeanalizować wszystkie możliwe przejścia między stanami

stabilnymi tablicy wtórnej przy wymuszeniu dozwolonych zmian wartości zmiennych wejściowych.

Jak łatwo się przekonać, wyścig krytyczny wystąpi przy przejściu z stanu (6) do (7) ($x_1x_2 = 01 \rightarrow 11$). (Wskaż inne rodzaje wyścigów).

Okazuje się, że kodowanie (b) także prowadzi do wyścigu krytycznego, natomiast przy kodowaniu (c) występuje tylko wyścig niekrytyczny, który w pewnych przypadkach jest do przyjęcia.

15.1.5 Eliminacja wyścigu krytycznego

Zauważmy, że jeżeli przekształcimy tablicę wtórną z rys. 15.9 do postaci podanych na rys. 15.10 (dla każdego kodowania osobno), to nie zmienimy zachowania się układu, a wyeliminujemy wyścig krytyczny w pierwszych dwóch tablicach. Wyjaśnienia wymaga tablica wtórna z rys. 15.10a.

kodowanie a)		kodowanie b)		kodowanie c)	
stan	y_1y_2	x_1x_2	y_1y_2	x_1x_2	y_1y_2
		00	01	11	10
3,2	00	1	4	(3)	(2)
1,6	01	(1)	(6)	5	2
4,7	10	1	(4)	(7)	2
5	11	1	—	7	—

Rys. 15.10 Eliminacja wyścigu krytycznego

Różnica między tablicą wtórną wyjściową a teraz rozważaną sprowadza się do zmian wprowadzonych w kolumnie $x_1x_2=11$, ponieważ występował tutaj wyścig krytyczny przy przejściu (6) \rightarrow (7) dla $x_1x_2 = 01 \rightarrow 11$. Eliminacja wyścigu polega na wymuszeniu chwilowego przejścia do wiersza 5, czyli zmianę $y_1 = 0 \rightarrow 1$ i dopiero w chwili następnej układ spowoduje zmianę $y_2 = 1 \rightarrow 0$. W rezultacie osiągamy stan stabilny (7). A zatem, do eliminacji wyścigu wykorzystaliśmy zmiany na stanach niestabilnych oraz (-).

Wszystkie przytoczone tablice wtórne mogą być wykorzystane do realizacji układu asynchronicznego. Okaze się jednak, że układy będą różne w sensie użytej liczby elementów logicznych.

a)

x_1x_2	00	01	11	10
(1)	7	—	4	
1	(3)	(2)	4	
5	(7)	2	—	
(5)	3	(6)	(4)	

b)

x_1x_2	00	01	11	10
(1)	2	(1)	(1)	
5	(2)	(2)	1	
(5)	7	(5)	1	
1	(7)	2	—	

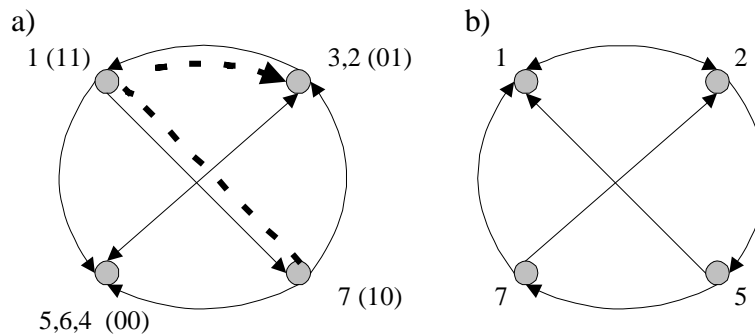
Rys. 15.11 Trudniejsze do zakodowania tablice wtórne

Przy kodowaniu tablicy wtórnej z niewielką liczbą wierszy pomocny jest graf przejść. Wierzchołki grafu odpowiadają wierszom tablicy, a strzałki wyznaczają przejścia międzywierszowe. Graf jest pomocny, ponieważ wierzchołki połączone strzałką powinny być kodowane tak, by wymuszać jednokrotną zmianę wartości zmiennych stanu. Rozważmy zatem nieco trudniejsze do zakodowania fikcyjne tablice wtórne, przedstawione na rys. 15.11 odpowiadające im grafy przejść z rys. 15.12. Obok oznaczenia wierszy wpisano propozycję zakodowania tablicy wtórnej. Łatwo zauważyć, że wyścig krytyczny może wystąpić przy przejściu (7) \rightarrow (2).

Aby pozbyć się wyścigu wystarczy oczywiście zastąpić stan niestabilny 2 w kolumnie 11 stanem niestabilnym 1 oraz kreskę stanem niestabilnym 2. Na rys. 15.12a zaznaczyliśmy ten fakt przerywanymi strzałkami. Zastosowaliśmy zatem znany

nam już sposób eliminacji wyścigu. Rozważając graf przejść z rys. 15.12b dochodzimy do wniosku, że żadne kodowanie nie eliminuje wyścigu krytycznego. Wyjściem z sytuacji jest wykorzystanie stanów pośrednich, co odpowiada wprowadzeniu dodatkowej pętli sprzężenia zwrotnego. Zakodowana tablica wtórna może wyglądać na przykład tak, jak to przedstawiono na rys. 15.13. Obok przedstawiliśmy odpowiadający jej graf przejść (hipersześcian).

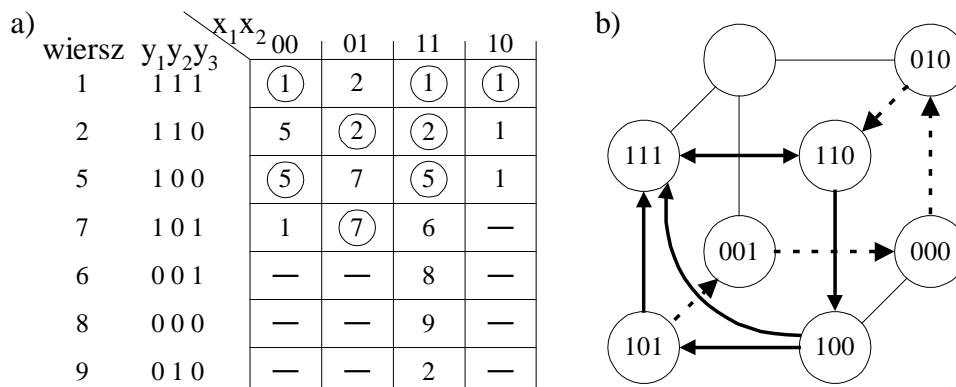
Wykorzystaliśmy zatem zmiany w stanach niestabilnych oraz (—). Wszystkie przytoczone tutaj tablice wtórne mogą być wykorzystane do realizacji układu asynchronicznego. Okazuje się jednak, że układy będą różne w sensie użytej liczby elementów logicznych.



Rys. 15.12 Grafy przejść

Podsumowując, możemy sformułować następujące spostrzeżenia:

1. W ogólnym przypadku tablicy z czterema (także trzema wierszami), w której istnieją przejścia pomiędzy wszystkimi stanami, nie można dobrać kodu tak, by przy każdym przejściu zmieniała się wartość wektora stanu na jednym sprzężeniu. Należy wprowadzić stany pośrednie, wykorzystując przejścia cykliczne (wyścig niekrytyczny).
2. Przed wykorzystaniem stanów pośrednich należy rozważyć możliwość wykorzystania stanów niestabilnych (—) do uzyskania przejścia cyklicznego eliminującego wyścig krytyczny.
3. W przypadku syntezy układu Moore'a staramy się tak kodować tablicę wtórną, aby wybrana zmienna stanu reprezentowała wprost zmienną wyjściową (omówimy to zagadnienie w następnych rozdziałach).



Rys. 15.13 Stany pośrednie

Tablicą przejść nazwiemy tablicę wtórną z wpisanymi wartościami zmiennych. Przypominając sobie definicję stanu stabilnego i niestabilnego nie powinniśmy mieć trudności z jej otrzymaniem (patrz rys. 15.14).

Pozostaje nam tylko otrzymać bezhazardową realizację układu, postępując odwrotnie niż przy analizie (patrz rozdział 14). Należy jedynie pamiętać, że tablica przejść musi być tablicą Karnaugh, na której dokonujemy minimalizacji funkcji przejść i eliminujemy potencjalną możliwość wystąpienia hazardu. Wiemy jednak, że przestawienie miejscami wierszy i kolumn tablicy przejść nie zmienia zachowania układu. Odpowiednie tablice Karnaugh przedstawiliśmy na rys. 15.15. Dla tablicy Karnaugh z rys. 15.15c otrzymamy funkcję przejść oraz realizację układu pokazaną na rys. 15.16.

Powróćmy do tablicy wtórnej z rys. 15.8. Biorąc pod uwagę kodowania z rys. 15.10c oraz permutację wierszy z rys. 15.14c otrzymamy zakodowaną tablicę wtórną pokazaną na rys. 15.17a.

a)	b)	c)
$ \begin{array}{c cccc} & \begin{array}{c} x_1 x_2 \\ \hline y_1 y_2 \end{array} & 00 & 01 & 11 & 10 \\ \hline 00 & 01 & 10 & 00 & 00 \\ 01 & 01 & 01 & 11 & 00 \\ 10 & 01 & 10 & 10 & 00 \\ 11 & 01 & — & 10 & — \end{array} $	$ \begin{array}{c cccc} & \begin{array}{c} x_1 x_2 \\ \hline y_1 y_2 \end{array} & 00 & 01 & 11 & 10 \\ \hline 00 & 01 & 10 & 00 & 00 \\ 01 & 01 & 01 & 11 & 00 \\ 11 & 01 & 11 & 11 & 00 \\ 10 & — & 11 & — & 00 \end{array} $	$ \begin{array}{c cccc} & \begin{array}{c} x_1 x_2 \\ \hline y_1 y_2 \end{array} & 00 & 01 & 11 & 10 \\ \hline 00 & 11 & 10 & 00 & 00 \\ 11 & 11 & 11 & 10 & 00 \\ 10 & 11 & 10 & 10 & 00 \\ 01 & 11 & — & — & 00 \end{array} $
$Y_1 Y_2$	$Y_1 Y_2$	$Y_1 Y_2$

Rys. 15.14 Tablice przejść dla tablic z rys. 15.10.

Na rys. 15.17b i c przedstawiliśmy wypełnione tablice wyjść dla układu Moore'a i Mealy'ego. Sposób otrzymania tablicy wyjść oraz funkcji wyjść $Z(y_1 y_2) = y_1 \bar{y}_2$ dla układu Moore'a nie powinien budzić wątpliwości. Zajmijmy się zatem otrzymywaniem tablicy wyjść dla układu Mealy'ego, realizując poniższy algorytm:

a)	b)	c)
$ \begin{array}{c cccc} & & \begin{array}{c} x_2 \\ \hline y_2 \end{array} & & \\ \hline & 01 & 10 & 00 & 00 \\ & 01 & 01 & 11 & 00 \\ \hline \begin{array}{c} y_1 \\ \hline \end{array} & 01 & — & 10 & — \\ & 01 & 10 & 10 & 00 \end{array} $	$ \begin{array}{c cccc} & & \begin{array}{c} x_2 \\ \hline y_2 \end{array} & & \\ \hline & 01 & 10 & 00 & 00 \\ & 01 & 01 & 11 & 00 \\ \hline \begin{array}{c} y_1 \\ \hline \end{array} & 01 & 11 & 11 & 00 \\ & — & 11 & — & 00 \end{array} $	$ \begin{array}{c cccc} & & \begin{array}{c} x_2 \\ \hline y_2 \end{array} & & \\ \hline & 11 & 10 & 00 & 00 \\ & 11 & — & — & 00 \\ \hline \begin{array}{c} y_1 \\ \hline \end{array} & 11 & 11 & 10 & 00 \\ & 11 & 10 & 10 & 00 \end{array} $
$x_1 \quad Y_1 Y_2$	$x_1 \quad Y_1 Y_2$	$x_1 \quad Y_1 Y_2$

Rys. 15.15 Tablice Karnaugh dla funkcji przejść z rys. 15.10

Krok 1. Stanom stabilnym przypisuje się wyjścia takie jak w pierwotnej tablicy Huffmana.

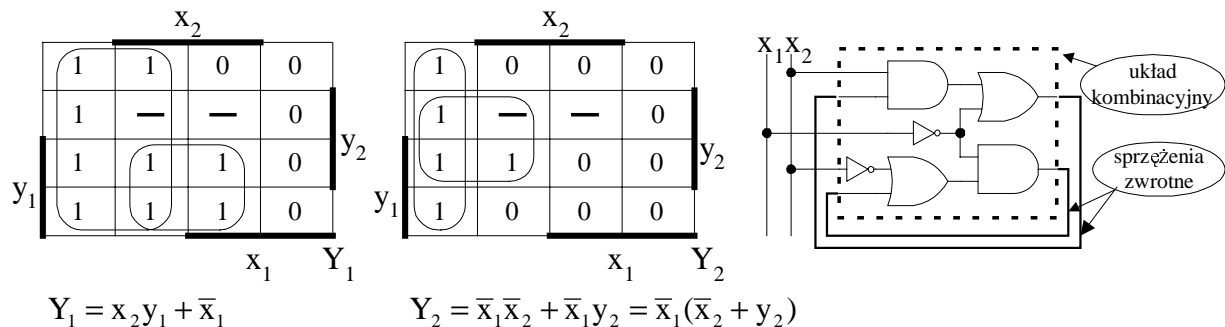
Krok 2. Stanom niestabilnym przypisuje się wyjścia zgodne z następującymi regułami:

- jeżeli sygnał z_i odpowiadający stanowi stabilnemu, z którego wychodzimy (stan początkowy) i stanowi stabilnemu, do którego dochodzimy (stan końcowy) są jednakowe, to taką samą wartość przypisujemy stanowi niestabilnemu,
- jeżeli sygnały z_i odpowiadające początkowemu i końcowemu stanowi stabilnemu są różne, to stanowi niestabilnemu przypisujemy sygnał dowolny (zaznaczony kreską).

Postępując zgodnie z algorytmem otrzymaliśmy tablicę wyjść z rys. 15.17c wraz z funkcją wyjść $Z(x_1 x_2 y_1 y_2) = x_2 y_1 \bar{y}_2$. W wybranych czterech miejscach tablicy wyjść wpisano duże litery. Zgodnie z tym algorytmem, w miejscu oznaczonym literką B można postawić kreskę (wpisano jednak zero). Wpisanie kreski umożliwia dowolny wybór wartości 0 albo 1, którego

dokonujemy w chwili wpisywania (minimalizacji) wyrażenia boolowskiego, obwodząc sąsiednie grupy jedynek i kresek zgodnie ze znanymi zasadami minimalizacji. Obwodząc kreskę decydujemy o wyborze wartości wyjścia równej jeden. Załóżmy, że tak właśnie zrobiliśmy. W naszym przypadku, tzn. przy przejściu (2)→(1) może się zdarzyć, że na moment układ przejdzie do stanu 10, a następnie 11 (uwarunkowane jest to odpowiednimi czasami opóźnień układu kombinacyjnego). Jeżeli wyjdziemy ze stanu (2) - wyjście $z=0$, gdy przebywamy w stanie 10 - wyjście $z=1$ i wreszcie w stanie 11 na wyjściu jest wartość 0.

Otrzymaliśmy tzw. migotanie wyjścia, które podobnie jak hazard może w nieokreślony sposób wpływać na układ sekwencyjny współpracujący z rozważanym układem.



Rys. 15.16 Układowa realizacja funkcji przejść

Analogiczne wyjaśnienia dotyczyć będą miejsc oznaczonych przez A, B, C i D. Oczywiście gdy układ nasz współpracuje z układem inercyjnym, który nie zdąży zareagować na chwilowe zmiany sygnału na wyjściu, wtedy stosujemy kreski zgodnie z algorytmem.

a) $x_1 x_2$

$y_1 y_2$	00	01	11	10	Z
00	1	4	(3)	(2)	0
01	1	—	—	2	—
11	(1)	(6)	7	2	0
10	1	(4)	(7)	2	1

b) $y_1 y_2$

Z
0 0
0 1
1 1
1 0

c) $x_1 x_2$

$y_1 y_2$	00	01	11	10
00	0	—	0	0
01	0 _A	—	—	0 _C
11	0	0	—	0
10	0 _B	1	1	0 _D

$Z(x_1 x_2 y_1 y_2)$

Rys. 15.17 Tablice wyjść: a - tablica wtórna, b - tablica wyjść dla układu Moore'a, c - tablica wyjść dla układu Mealy'ego

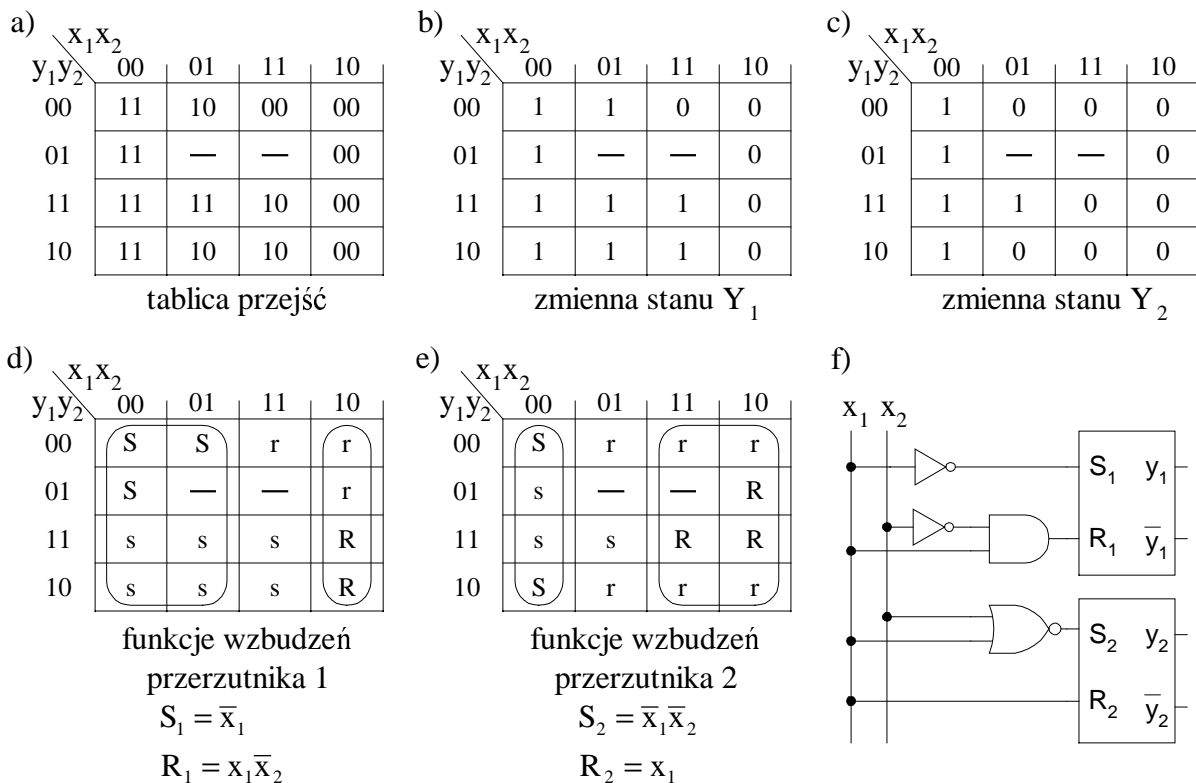
15.2 Synteza układu z wykorzystaniem przerzutnika asynchronicznego

Analizą układu asynchronicznego z wyodrębnionym blokiem przerzutników asynchronicznych typu SR zajmowaliśmy się w podrozdziale 14.3. Obecnie naszym celem jest synteza układów kombinacyjnych wzbudzających wejścia S i R przerzutników tak, by zestaw wartości zmiennych stanu (wyjść przerzutników) zmieniał swoje wartości zgodnie z tablicą przejść. Stany stabilne wskazują te miejsca tablicy przejść, dla których nie wymaga się żadnej zmiany stanu przerzutników. Stany niestabilne wskazują na wymaganą zmianę wartości odpowiedniej zmiennej stanu.

Jeżeli wymagana jest zmiana $0 \rightarrow 1$, należy na przerzutnik reprezentujący tę zmienną stanu podać na wejście S wartość 1 (przy $R=0$), a przy zmianie $1 \rightarrow 0$ na odwrót - $R=1$ ($S=0$). Z wiadomego już powodu należy pamiętać, że nie wolno dopuścić do sytuacji $S=R=1$. Aby nie

powtarzać rozumowania przeprowadzonego w p.14.3, od razu podamy kolejne kroki algorytmu umożliwiającego syntezę układu:

1. Narysuj tablice Karnaugh'a dla każdej zmiennej stanu y_i z osobna,
2. Przepisz z tablicy przejść wartość zmiennej stanu do odpowiadającej jej tablicy Karnaugh'a,
3. W tablicy Karnaugh'a odpowiadającej zmiennej y_i zamień wartość 1(0) na s(R), gdy $y_i=1$ oraz wartość 1(0) na S (r), gdy $y_i=0$,
4. Dokonaj minimalizacji funkcji wzbudzającej wejścia S i R i - tego przerzutnika, traktując S i R jako iloczyny zupełne obowiązkowe, zaś r i s oraz kreskę jako iloczyny obojętne (przy minimalizacji funkcji przełączającej, w grupie obwiedzionych symboli musi wystąpić przynajmniej jedno S albo R - dlaczego?),
5. Narysuj układ.

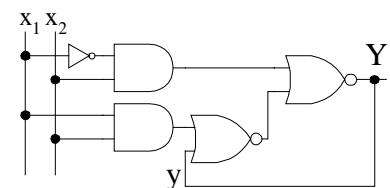


Rys. 15.18 Synteza układu asynchronicznego na przerzutnikach asynchronicznych SR:
a,b,c - krok 1 i 2 algorytmu, d,e - kroki 3 i 4, f - schemat układu

Na rys. 15.18 pokazaliśmy poszczególne kroki algorytmu na przykładzie z poprzedniego podrozdziału (rys. 15.17). Syntezy układu wyjść dokonujemy w sposób analogiczny do stosowanego przy układach ze sprzężeniem zwrotnym.

Weźmy pod uwagę układ asynchroniczny podany na rys. 15.19 i porównajmy go z układem przedstawionym na rys. 14.15a, który wykorzystaliśmy do wyjaśnienia wpływu zjawiska hazardu na zachowanie się układu. Przypomnijmy, że układ z rys. 14.15a może zachowywać się niezgodnie z opisującą go tablicą Huffmana (rys. 14.15c).

Funkcja przejść dla układu z rys. 15.19 jest następująca: $Y(x_1x_2y) = x_1x_2 + x_1y + \bar{x}_2y$.

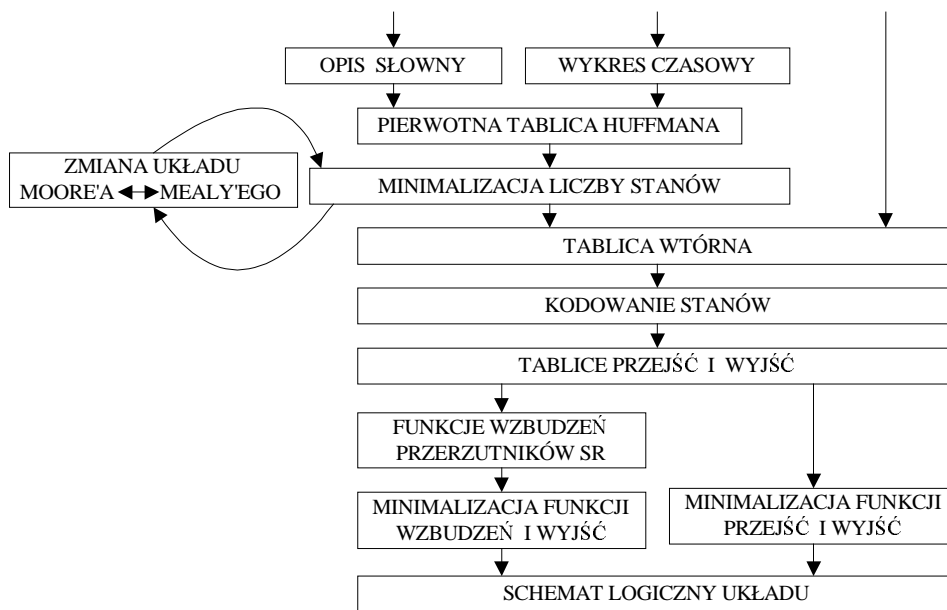


Rys. 15.19 Bezhazardowa realizacja układu asynchronicznego

Nietrudno zauważyć, że jest to ta sama funkcja przejść, która opisuje układ z rys. 14.15a, z tą różnicą, że powyższa jest sumą wszystkich implikantów prostych. Zatem, zgodnie z twierdzeniem 14.6.1, realizacja ta jest bezhazardowa. Jest to główna przesłanka przemawiająca za stosowaniem w układach asynchronicznych przerzutnika SR (asynchronicznego).

15.3 Podsumowanie algorytmu syntezy układów asynchronicznych

Projektowanie układów asynchronicznych nie jest proste ze względu na konieczność uwzględniania omówionych w poprzednich rozdziałach szkodliwych zjawisk, jakimi są hazard i wyścig krytyczny. Rozważania nasze podsumujemy przypomnieniem metody Huffmana w zwartej formie (rys. 15.20) oraz przykładem syntezy wybranego układu, którego zachowanie zadamy pierwotną tablicą Huffmana (rys. 15.21).



Rys. 15.20 Etapy syntezy układów asynchronicznych

Syntezerowany układ posiada trzy wejścia x_1 , x_2 , x_3 oraz jedno wyjście Z . Zakładamy, że mamy otrzymać układ Moore'a (tablica wtórna z rys. 15.21b powstała przy tym założeniu). Po zakodowaniu tablicy wtórnej, otrzymujemy tablice przejść (wyścig niekrytyczny występuje tylko w kolumnie 010 oraz 100). Tablicę przejść z rys. 15.21b przedstawiliśmy w postaci tablicy Karnaugh'a (zamieniliśmy miejscami wybrane wiersze). Funkcje przejść są następujące:

$$Y_1 = x_2 + x_1 x_3 y_1 + x_1 x_3 \bar{y}_2$$

$$Y_2 = x_2 + \bar{x}_1 x_3 + \bar{y}_1 y_2 + \bar{x}_1 y_2 + x_1 \bar{x}_3 y_1$$

a funkcja wyjść postaci - $Z=y_1$. Jako ćwiczenie należy narysować schemat układu.

15.4 Problemy syntezy układów asynchronicznych

Czytelnik po zapoznaniu się z treścią poprzednich rozdziałów mógłby stwierdzić, że konstruowane przez niego układy asynchroniczne będą działać poprawnie. Nic bardziej mylnego, zakładając nawet, że poprawnie je zsyntezował, użył sprawnych elementów i odpowiednio je połączył.

Przypomnijmy sobie założenia (w rozszerzonej postaci) na podstawowy tryb pracy (*fundamental mode*) układu asynchronicznego:

1. Tylko jeden sygnał wejściowy ulega zmianie w danej chwili. Kolejna zmiana może wystąpić, gdy układ znajduje się w stanie stabilnym.
2. Układ nie ma wyścigu krytycznego.
3. Układ przejść i wyjść nie ma hazardu statycznego (dynamicznego),
4. Maksymalne opóźnienie zmiany sygnału poprzez układ przejść i pętle sprzężenia zwrotnego jest mniejsze niż przedział czasu pomiędzy kolejnymi zmianami sygnału wejściowego.
5. Minimalne opóźnienie wprowadzane przez układ przejść i pętle sprzężenia zwrotnego jest większe niż maksymalny czas asynchronizmu wejść.

a)

$x_1x_2x_3$	000	001	011	010	110	111	101	100	Z
①	5	—	3	—	—	2	—	0	
1	—	—	—	—	—	②	9	1	
4	—	—	③	7	—	—	—	1	
④	—	—	3	—	—	8	—	0	
4	⑤	6	—	—	—	—	—	0	
—	5	⑥	3	—	—	—	—	1	
—	—	—	3	⑦	—	2	—	1	
4	—	—	—	—	—	⑧	—	0	
—	5	—	—	—	—	—	⑨	0	

b)

$x_1x_2x_3$	000	001	011	010	110	111	101	100	Z
00	①	5	—	3	—	—	2	—	0
11	4	5	⑥	③	⑦	—	2	—	1
10	1	—	—	—	—	—	②	9	1
01	④	⑤	6	3	—	—	⑧	⑨	0

c)

x_1	00	01	—	11	—	—	10	—
y_2	01	01	11	11	—	—	01	01
y_1	01	01	11	11	11	—	10	—
Y_1Y_2	00	—	—	—	—	—	10	01

d)

x_1	0	0	—	1	—	—	1	—
y_2	0	0	1	1	—	—	0	0
y_1	0	0	1	1	1	—	1	—
Y_1	0	—	—	—	—	—	1	0

e)

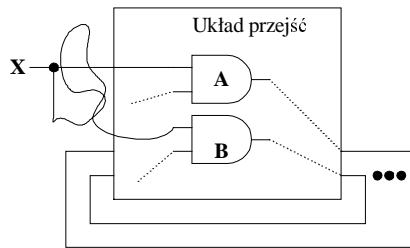
x_1	0	1	—	1	—	—	0	—
y_2	1	1	1	1	—	—	1	1
y_1	1	1	1	1	1	—	0	—
Y_2	0	—	—	—	—	—	0	1

Rys. 15.21 a - pierwotna tablica Huffmana, b - wtórna tablica Huffmana (układ Moore'a), c - tablica przejść dla funkcji Y_1 i Y_2 , d - tablica przejść dla funkcji Y_1 , e - tablica przejść dla funkcji Y_2

Założenie (1) musi być spełnione, aby układ działał poprawnie, ponieważ układ powinien mieć czas na przejście do następnego stanu stabilnego. Spełnienie założeń (2) i (3) sprawi, że układ działa poprawnie, co dosyć szczegółowo omówiliśmy w poprzednich rozdziałach. Założenie (4) to warunek na spełnienie założenia (1).

Można się spodziewać, że zagadnienie o którym chcemy tutaj powiedzieć wynika z założenia (5). Jeżeli spełnione będą pierwsze cztery założenia a ostatnie nie, to układ nie będzie działał poprawnie, co ujawnia się przejściem układu do stanu stabilnego niezgodnego ze specyfikacją. Zmiana sygnału wejściowego, która propaguje się do układu przejść nie dotarła jeszcze do niektórych jego wejść (czas asynchronizmu) ale ulega zmianie wartość pewnej zmiennej stanu, która poprzez sprzężenie zwrotne propaguje się na wejścia układu przejść. Jest to konsekwencja niespełnienia założenia (5), ponieważ musi istnieć w układzie ścieżka z tak małym sumarycznym opóźnieniem, że zmiana sygnału zdążyła się przenieść na wyjście układu przejść zmieniając wartość zmiennej stanu.

Założmy, że zmiana sygnału X dociera do wejścia bramki A w chwili $t=1$ (rys. 15.22), dalej propaguje się poprzez układ przejść do wyjścia i poprzez sprzężenie zwrotne dociera na jego wejście (chwila $t=2$). Dopiero w chwili $t=3$ ta sama zmiana sygnału x dociera do wejścia bramki B (co zaznaczyliśmy wyolbrzymieniem długości przewodu), propaguje się do wyjścia powodując zmianę wartości zmiennej stanu niezgodną ze specyfikacją. W rozważanym przypadku czas asynchronizmu wejść wynosi dwie jednostki czasu.



Rys. 15.22 Hazard istotny

Jedynym sposobem uniknięcia tego zjawiska, nazywanego hazardem istotnym (*essential hazard*), jest zapewnienie by zmiana sygnału wejściowego dotarła do wszystkich wejść układu, zanim wystąpi jakakolwiek zmiana wartości zmiennych stanu, czyli należy wprowadzić dodatkowe opóźnienia na najszybszych ścieżkach od wejść do wyjść.

Wykrycie hazardu istotnego jest możliwe ale nie we wszystkich układach. Występuje on, gdy w tablicy Huffmana znajdziemy taki stan stabilny Y i zmienną wejściową X , której trzy kolejne zmiany wartości doprowadzają układ do stanu stabilnego różnego od stanu, gdy X zmienimy tylko raz (w niektórych źródłach jest to definicja tego hazardu). Zatem wystąpienie hazardu istotnego jest możliwe tylko w układach, które mają co najmniej trzy stany stabilne, a jego nazwa wzięła się stąd, że jest on „istotny” dla tablicy Huffmana bez względu na realizację funkcji przejść i może być wyeliminowany tylko poprzez wprowadzenie opóźnień.

ĆWICZENIA

- Czy podany zestaw cech jest charakterystyczny dla pierwotnej tablicy Huffmana:
 - w każdym wierszu występuje co najmniej jedna kreska dotycząca przejść,
 - w każdym wierszu występuje tylko jeden stan stabilny,
 - opisuje układ typu Moore’a,
 - określony stan niestabilny występuje tylko w tej kolumnie, w której występuje odpowiadający mu stan stabilny.
- Dla wykresu skracania z rys. 15.7c zaproponuj postać zredukowanej tablicy pierwotnej.
- Dla podanych na rys. 15.23 tablic Huffmana podaj realizacje układowe.
- Sformułuj opis słowny zachowania się układów z rys. 15.24.

x_1x_2	00	01	11	10	Z_1Z_2
①	2	—	4	00	
1	⑦	3	—	10	
—	2	⑥	4	01	
5	②	3	—	01	
—	2	③	4	10	
⑤	7	—	4	00	
1	—	6	④	11	

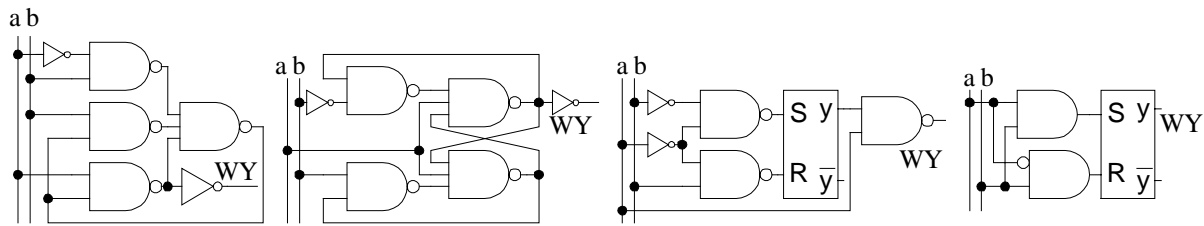
x_1x_2	00	01	11	10	Z
①	—	—	2	0	
⑤	—	—	6	0	
1	—	7	⑥	1	
—	4	⑦	—	1	
—	—	3	②	0	
—	4	③	—	0	
5	④	—	—	0	

x_1x_2	00	01	11	10	Z
①	3	—	2	0	
4	③	—	—	0	
④	6	—	5	1	
4	—	—	②	0	
1	⑥	—	—	1	
1	—	—	⑤	1	

Rys. 15.23 Przykładowe tablice Huffmana

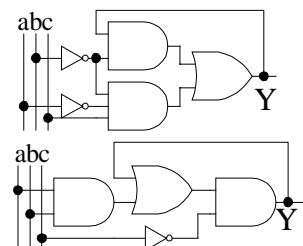
- Dla układów z rys. 15.25 podaj wtórne tablice Huffmana oraz wpisz funkcje Y do tablicy Karnaugh.
- Zaproponuj układ asynchroniczny rozpoznający położenie przedmiotu na taśmociągu (rys. 15.26). Fotokomórka zasłonięta daje sygnał 0. Dane są tylko dwuwejściowe bramki NAND.

7. Zsyntezuj układ asynchroniczny dwuwejściowy (a,b) i jednowyjściowy (z), w którym na wyjściu ukazuje się jedynka tylko wtedy, gdy zmienna wejściowa a zmienia się z jedynki na zero bądź zmienna wejściowa b zmienia się z zera na jedynkę.



Rys. 15.24 Przykładowe układy asynchroniczne

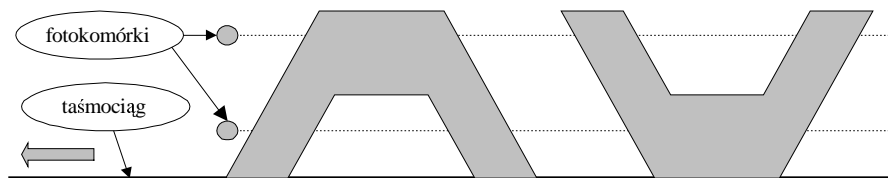
8. Zsyntezuj układ asynchroniczny dwuwejściowy (a,b) i jednowyjściowy (z), w którym jedynka na wyjściu pokazuje się tylko wtedy, gdy zmienna wejściowa a zmienia się z zera na jedynkę, bądź zmienna wejściowa b zmienia się z jedynki na zero. Porównaj rozwiązanie z rozwiązaniem zadania 7.



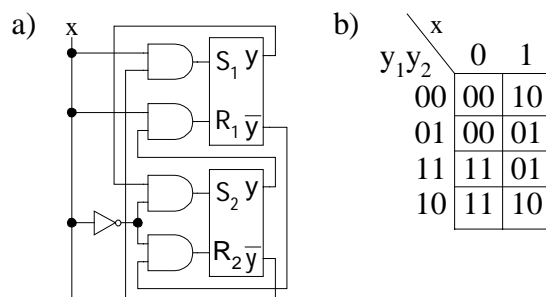
Rys. 15.25 Przykładowe układy asynchroniczne

9. W układzie z rys. 15.27 nie występuje zjawisko wyścigu ani hazardu. Dlaczego jednak po zrealizowaniu układu istnieje duże prawdopodobieństwo, że w przypadku zmiany $x = 0 \rightarrow 1$ układ przejdzie do stanu 01, zamiast do stanu 10 (stanem początkowym jest stan 00). W jaki sposób wyeliminować opisane zjawisko?

10. Zsyntezuj układ asynchroniczny o dwóch wejściach a i b oraz jednym wyjściu Z. Po włączeniu zasilania $Z=0$. Jeżeli wartość wejścia a zmienimy z 0 na 1 to zawsze $Z=1$ oraz Z zmienia swoją wartość z 1 na 0 gdy $b=1$.



Rys. 15.26 Taśmociąg i położenie fotokomórek



Rys. 15.27 Przykładowy układ asynchroniczny: a - schemat, b - tablica Huffmana

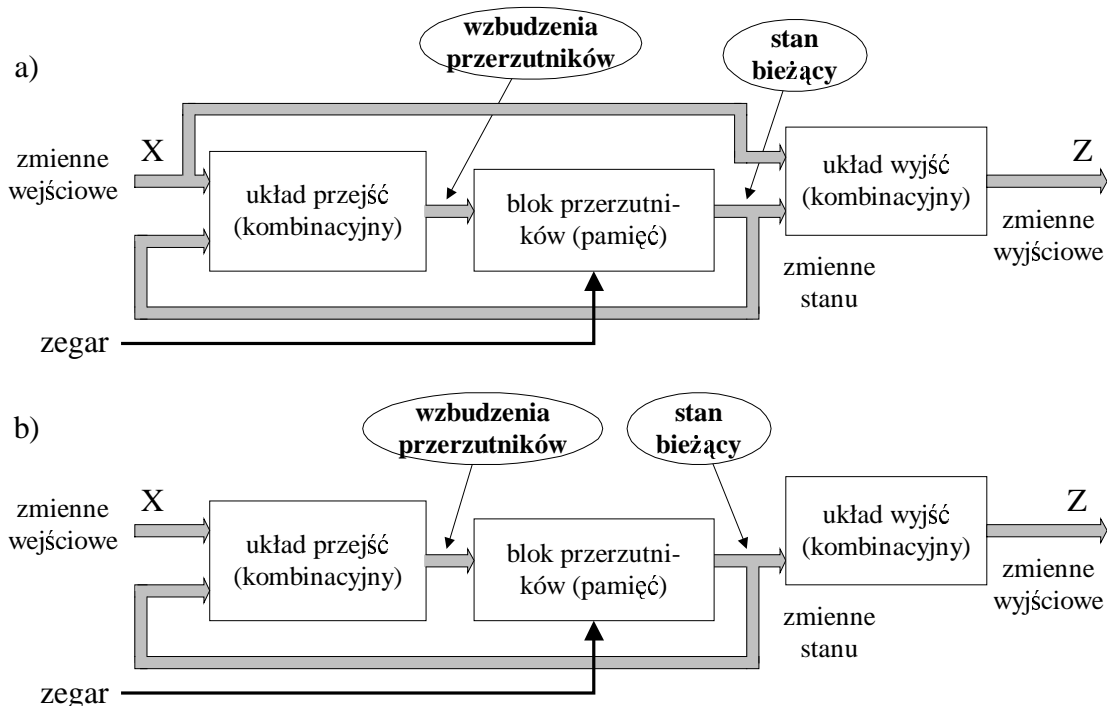
16 UKŁADY SYNCHRONICZNE

W poprzednich rozdziałach zajmowaliśmy się układami kombinacyjnymi oraz sekwencyjnymi układami asynchronicznymi. W tym i następnym rozdziale zajmiemy się projektowaniem układów synchronicznych, które tak jak układy asynchroniczne, należą do klasy układów sekwencyjnych.

Przy projektowaniu układów synchronicznych mogą być wykorzystane metody stosowane w projektowaniu układów asynchronicznych. Należy oczywiście uwzględnić kilka podstawowych różnic między tymi układami. Do ważniejszych należą następujące:

1. Układ synchroniczny jest układem przetwarzającym sygnały wejściowe, jeżeli pewien wyróżniony w układzie sygnał (nazywany zegarowym), przyjmie wartość wyróżnioną. Sygnał zegarowy nazywamy także synchronizującym lub taktującym, ponieważ narzuca on rytm pracy układu. Stąd chwile pojawienia się wyróżnionej wartości sygnału zegarowego będziemy przyjmować za umowne jednostki czasu i oznaczać zazwyczaj przez t , $t+1$, $t+2$...
2. Wartości sygnałów wejściowych ulegają zmianie, pomiędzy wyróżnionymi wartościami sygnału zegarowego.
3. W układach synchronicznych wszystkie stany wewnętrzne układu są stanami stabilnymi (nie występuje zjawisko wyścigu).

Pozostałe aspekty procesu projektowania układu, takie jak tablica stanów-wyjść czy minimalizacja liczby stanów, praktycznie pozostają bez zmian.

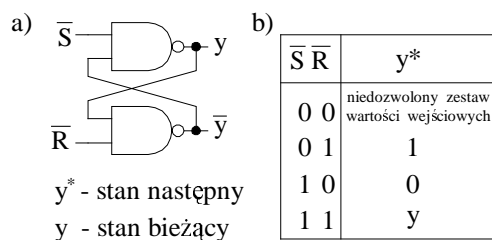


Rys. 16.1 Blokowa struktura układów synchronicznych: a - układ Mealy'ego, b - układ Moore'a

Strukturę blokową układów synchronicznych przedstawiliśmy na rys.16.1. Podział układów na układy Mealy'ego [1] i Moore'a [2] jest identyczny z podziałem dokonany dla układów asynchronicznych.

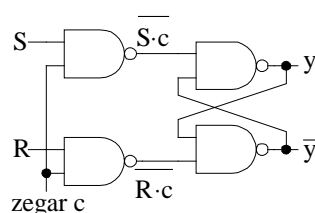
16.1 Przerzutniki synchroniczne

Układy synchroniczne należą do klasy układów sekwencyjnych, czyli układów pamiętających historię swoich wejść. Tym samym w skład układu muszą wchodzić elementy umożliwiające zapamiętanie tej historii. Wykorzystywanymi w tym celu elementami pamiętającymi są przerzutniki synchroniczne. Są to dwustanowe synchroniczne elementy, zmieniające stan w chwili wystąpienia sygnału taktującego na wejściu zegarowym. Stan przerzutnika zależy od wartości na jego wejściach informacyjnych (wzbudzających). Wyróżnia się kilka rodzajów przerzutników synchronicznych, różniących się między sobą liczbą wejść wzbudzających (zazwyczaj jedno albo dwa) oraz funkcją wzbudzeń, tzn. zależnością między wartościami sygnałów wzbudzających, a stanem przerzutnika po podaniu sygnału taktującego. Rola, jaką mają do spełnienia przerzutniki synchroniczne, jest bardzo ważna z punktu widzenia poprawności działania całego układu synchronicznego, dlatego omówimy je dosyć szczegółowo.



Rys. 16.2 Przerzutnik asynchroniczny:
 a - układ NAND, b - zasada działania

Rozważając układy asynchroniczne wprowadziliśmy pojęcie elementarnego układu pamiętającego, który nazwaliśmy przerzutnikiem asynchronicznym SR. Był to prosty układ, składający się z dwóch połączonych na krzyż bramek NOR. Pozwalał na zapamiętanie wartości 1 albo 0 na wyjściu i utrzymanie jej przez praktycznie dowolnie długi czas, aż do momentu żądania zapamiętania przeciwnej wartości logicznej. Przerzutniki synchroniczne wykorzystują przerzutniki asynchroniczne SR wykonane z bramek NAND (Rys. 16.2). Szczególną uwagę zwróćmy na różnice w oznaczeniach i działaniu NAND-owego układu, w porównaniu z układem złożonym z bramek NOR. Analizując tabelkę z rys. 16.2b przekonujemy się, że zasada działania przerzutnika złożonego z bramek NAND jest analogiczna do przerzutnika złożonego z bramek NOR, ponieważ wartość wyjścia przerzutnika nie ulega zmianie, gdy $\bar{S} = \bar{R} = 1$, oraz przyjmuje wartość 1 (0), gdy $\bar{S} = 0, \bar{R} = 1$ ($\bar{S} = 1, \bar{R} = 0$). Gdy $\bar{S} = \bar{R} = 0$, działanie przerzutnika jest nieokreślone z powodów analogicznych do tych, które omówiliśmy w rozdziale poświęconym układom asynchronicznym.



Rys. 16.3 Synchroniczny
 przerzutnik SR

Synchroniczny przerzutnik SR możemy zbudować z przerzutnika asynchronicznego dokładając do niego dwie bramki NAND, tak jak to pokazaliśmy na rys.16.3 (zwróćmy uwagę na zmianę oznaczeń). Dodatkowe bramki NAND wykorzystuje się do zablokowania wejść SR na czas, w którym impuls zegara $c=0$. Gdy $c=0$, wówczas wejście $\bar{S}c = \bar{R}c = 1$, zatem przerzutnik nie zmienia wartości sygnałów swoich wyjść. Jeżeli na wejście c podamy wartość 1, wartości sygnałów S i R po dopełnieniu na bramkach NAND podawane są bezpośrednio na przerzutnik asynchroniczny. Wartość jego wyjścia będzie teraz zależeć od wartości S i R , zgodnie z tabelką, którą przedstawiliśmy na rys.16.2b.

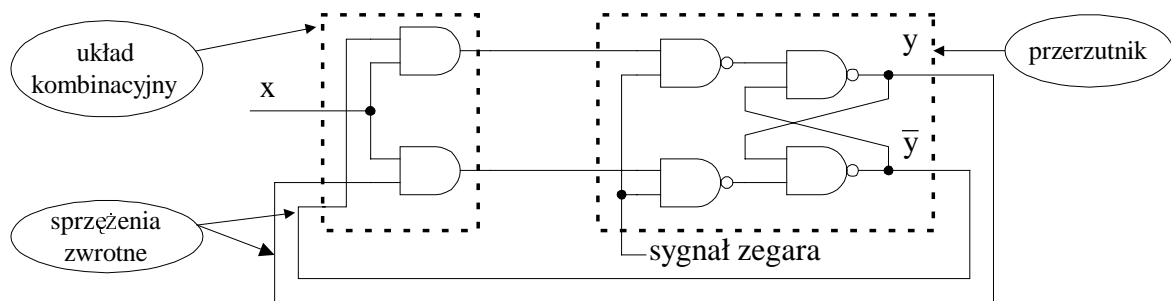
Zauważmy, że jeżeli w pewnym odpowiednio długim przedziale czasu, wartość sygnału taktującego $c=1$ oraz wartości sygnałów S , R ulegną kilkakrotnym zmianom, to przerzutnik odnowi swój stan zgodnie z nowymi sygnałami wejściowymi, czyli zachowa się jak przerzutnik asynchroniczny. Jak dalej pokażemy, powyższe zachowanie przerzutnika synchronicznego ma raczej niekorzystny wpływ na poprawne działanie układu synchronicznego. W celu zilustrowania tego wpływu rozważymy układ z rys. 16.4.

Dowolny układ synchroniczny, a więc i ten z rys.16.4, przy ustalonych wartościach

sygnału wejściowego x powinien zmienić swój stan tylko jeden raz po podaniu wyróżnionej wartości sygnału taktującego.

W rozważanym układzie wartość sygnału wyjściowego przerzutnika zmienia się $0 \rightarrow 1 \rightarrow 0 \rightarrow 1 \rightarrow 0$, gdy sygnał taktujący przyjmie wartość $c=1$, ponieważ sygnały wyjściowe przerzutnika reagują na zmiany wartości sygnałów S i R gdy $c=1$, a te z kolei zależą od sygnałów wyjściowych przerzutnika poprzez bramki AND.

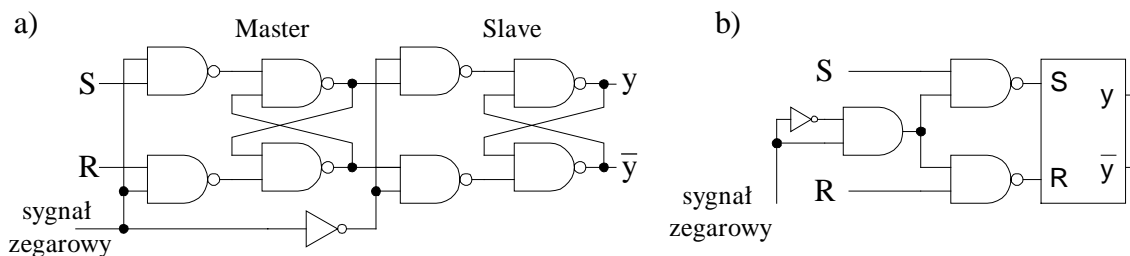
Układ będzie działał poprawnie, jeżeli czas trwania sygnału zegarowego nie będzie większy od czasu opóźnienia wyjściowych sygnałów przerzutnika (co praktycznie sprowadza się do wprowadzenia opóźnień na wyjściu przerzutnika). Wtedy przerzutnik w czasie trwania sygnału taktującego zmieni swój stan tylko jeden raz. Takie rozwiązanie ma szereg wad. Po pierwsze, ulega zmniejszeniu szybkość działania całego układu. Po drugie, źródło sygnałów taktujących (zegar) powinno generować sygnały o szerokości nie większej od opóźnień sygnałów od wejść do wyjść przerzutnika oraz nie mniejszej od czasu, w którym przerzutnik zdolny jest zareagować na wartości wzbudzających go sygnałów wejściowych.



Rys. 16.4 Przykładowy układ synchroniczny

W celu uniknięcia jednostkowego strojenia (kontrolowania czasu propagacji) każdego układu synchronicznego stosuje się dwa różne podejścia, sprowadzające się do odpowiedniego zaprojektowania przerzutnika.

W pierwszym podejściu stosuje się przerzutniki wyzwalane zbochem (*edge-triggered flip-flop*). Zmiana stanu przerzutnika następuje tylko w trakcie trwania zbocza impulsu zegarowego przedniego (wznoszącego), gdy $c=0 \rightarrow 1$, albo tylnego (opadającego), gdy $c=1 \rightarrow 0$. Po jego zakończeniu przerzutnik nie jest już czuły na zmiany wartości sygnałów wejść informacyjnych. Prosta realizację takiego przerzutnika pokazaliśmy na rys.16.5b (jaki zjawisko tutaj wykorzystano?). Przerzutniki tego typu są jednak czułe na czas trwania zbocza sygnału zegarowego.

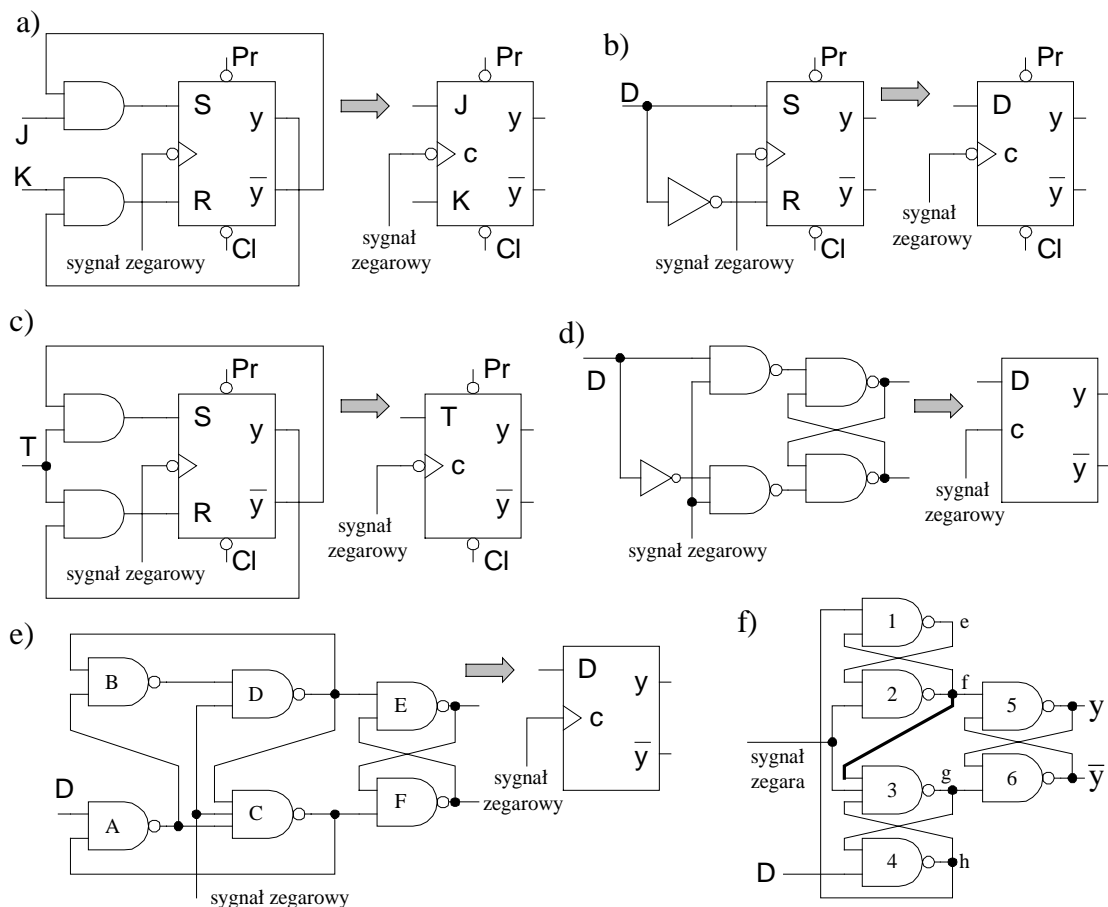


Rys. 16.5 Przerzutniki synchroniczne: a - przerzutnik dwutaktowy, b - przerzutnik wyzwalany zbochem

W drugim podejściu stosuje się przerzutniki wyzwalane dwustopniowo. Rozważmy układ z rys. 16.5a, nazywany przerzutnikiem dwutaktowym (*master-slave*), składający się z kaskadowego połączenia dwóch przerzutników synchronicznych SR. Jeżeli sygnał taktujący

zmienia się z zera na jeden (zbrocze wznoszące), w pierwszej kolejności nastąpi zablokowanie drugiego przerzutnika (*slave*) od wyjść pierwszego (*master*). Przerzutnik master odnowi swój stan zgodnie z wejściami S i R. Jeżeli teraz sygnał taktujący zmieni się z jeden na zero (zbrocze opadające), to nastąpi zablokowanie pierwszego przerzutnika od jego wejść informacyjnych oraz odblokowanie wejść przerzutnika drugiego, czyli przypisanie stanu przerzutnika master do przerzutnika slave. Ten typ przerzutnika może działać poprawnie bez względu na czas trwania zboczy sygnału taktującego. Sygnałem taktującym może być nawet ogólnie dostępny sygnał sinusoidalny.

Wyróżnia się kilka typów przerzutników synchronicznych, jak JK, D oraz T. Przerzutniki te otrzymujemy z dwutaktowych przerzutników SR, umieszczając w układzie niewielką liczbę dodatkowych bramek (rys.16.6).



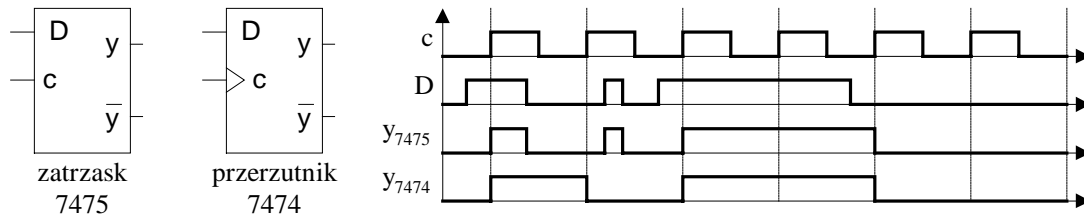
Rys.16. 6 Przykłady przerzutników: a - JK, b - D, c - T, d - zatrząsk, e,f - D wyzwalany przednim zboczem

W produkowanych przerzutnikach udostępnia się dodatkowe wejścia, zazwyczaj oznaczane Pr (*preset*) oraz Cl (*clear*). Wejścia te nazywa się asynchronicznymi, ponieważ zmiana wartości sygnałów na ich wejściach powoduje natychmiastową i niezależną od sygnału taktującego zmianę stanu przerzutnika. Służą one do początkowego ustawienia przerzutnika w stan $y=1$ (Pr=0, Cl=1), bądź $y=0$ (Pr=1, Cl=0). Jeżeli Pr=Cl=1, to przerzutnik działa w trybie synchronicznym i jego stan zależy od stanu wejść informacyjnych i sygnału zegarowego.

W katalogach elementów cyfrowych możemy spotkać układy, które nazywamy zatrząskiem (*latch*, w przeciwieństwie do *flip-flop* - przerzutnik), bardzo często utożsamiane z przerzutnikiem synchronicznym typu D. Jest to element wyzwalany poziomem, czyli reagujący na zmiany sygnału na wejściu informacyjnym przez cały czas trwania impulsu

zegarowego ($c=1$). Zatraski stosuje się jedynie do buforowania informacji podczas przekazywania jej z określonego źródła (ich zespół nazywamy buforem). Strukturę wewnętrzną zatrasku oraz przerzutnika D wyzwalanego przednim zboczem przedstawiliśmy na rys. 16.6d i e, aby Czytelnik mógł szczegółowo przeanalizować działanie obu układów (zwróćmy uwagę na różnice występujące przy wejściu zegarowym).

W analizie może pomóc rys. 16.7, na którym pokazaliśmy przebiegi czasowe zatrasku (7475) oraz typowego przerzutnika reagującego na zbocze wznoszące (element 7474).



Rys. 16.7 Porównanie zachowań zatrasku i przerzutnika D

Porównując rys. 16.6e i f zauważymy, że układy tam przedstawione różnią się jedynie sposobem rozmieszczenia elementów (proponujemy przyporządkowanie bramek oznaczonych literkami do odpowiednich bramek oznaczonych cyferkami). Poprzez inne narysowanie układu chcieliśmy zwrócić uwagę na pewne podobieństwo z układem rys. 16.6d. Otóż nietrudno zauważyć, że wejściowe bramki NAND zostały zastąpione przez asynchroniczne przerzutniki SR (bramki 1-2 i 3-4). Zamiast bramki NOT wprowadzono połączenie między wyjściem bramki 2 i wejściem bramki 3 (na rysunku zostało ono pogrubione). Jest ono konieczne, ponieważ pozwala uniknąć jednoczesnego przekazania na wejścia bramek 5 i 6 wartości 0 (dlaczego unikamy tej sytuacji?).

Celem poniższej analizy układu z rys. 16.6f nie będzie pokazanie, że układ jest przerzutnikiem D reagującym tylko przy narastającym zboczu zegara, lecz przede wszystkim ma na celu określenie dwóch bardzo ważnych parametrów czasowych. Aby ułatwić analizę układu, każdą bramkę ponumerowaliśmy od 1 do 6 a wyjścia wybranych czterech bramek oznaczyliśmy literkami e, f, g, h. Przyjmijmy także, że każda bramka NAND występująca w układzie wnosi takie samo jednostkowe opóźnienie, co umożliwi nam reprezentowanie czasu w postaci liczb całkowitych. Oznaczmy sygnał zegarowy literką C.

Rozpocznijmy analizę układu w chwili $t=0$ od stanu określonego następującymi wielkościami: $C=0$, $D=0$ oraz wyjścia $f=g=h=1$, $e=0$ oraz $y=1$. Jeżeli w chwili $t=1$ wartość sygnału zegarowego C ulegnie zmianie na 1, to bramki do których ta zmiana dotrze zareagują dopiero po upływie czasu opóźnienia, czyli w chwili $t=2$ zmienna g przyjmie wartość równą 0. To z kolei spowoduje po upływie jednego opóźnienia ($t=3$) zmianę wyjścia $\bar{y} = 0 \rightarrow 1$, a w chwili $t=4$ wyjście $y = 1 \rightarrow 0$ (czyli między chwilą $t=3$ a $t=4$ $y = \bar{y} = 1$). Zatem wartość $D=0$ została przeniesiona na wyjście y po upływie czasu równym trzem jednostkowym opóźnieniom licząc od chwili $t=1$, gdy sygnał zegara zmienił swoją wartość z 0 na 1.

Zauważmy, że jeżeli tylko zmienna g przyjmie wartość 0 (bo $D=0$) w chwili $t=2$, to od tego momentu wartość D może ulec zmianie, bez wpływu na wartości pozostałych zmiennych układu. Co by się stało, gdyby D zmieniło swoją wartość zanim upłynąłby czas równy jednemu opóźnieniu? Działanie przerzutnika asynchronicznego SR byłoby nieokreślone.

Zatem poprawne działanie przerzutnika wymaga by po zmianie sygnału zegarowego z 0 na 1, sygnał na wejściu D nie ulegał zmianie przez czas równy jednemu opóźnieniu. Dlatego czas ten nazywamy czasem utrzymania informacji (*hold time*) po wznoszącym zboczu zegara.

Rozpocznijmy jeszcze raz analizę układu w chwili $t=i$ od stanu określonego następującymi wielkościami: $C=0$, $D=0$ oraz wyjścia $f=g=h=1$, $e=0$ oraz $y=0$. Jeżeli zmienimy w chwili $t=i+1$ wartość zmiennej D z 0 na 1 (sygnał zegarowy C nadal pozostaje 0), to w chwili $t=i+2$ zmianie ulegnie wyjście $h=1 \rightarrow 0$, które pobudzi bramkę 1. W rezultacie, w chwili $t=i+3$ wyjście e tej bramki zmieni wartość z 0 na 1. Zmiana ta z kolei przeniesie się na wejście bramki 2, lecz nie ulegnie zmianie wartość $f=1 \rightarrow 1$ (zmiana sygnału przestała się propagować oraz wartość y nie uległa zmianie). Zatem po zmianie wartości wejścia D należy odczekać czas równy trzem opóźnieniom, ponieważ tyle trwa ustalenie się wartości wyjść h oraz e .

Zatem po dowolnej zmianie wartości wejścia D , należy odczekać, w rozważanym przez nas przypadku, przynajmniej czas równy opóźnieniom trzech bramek, by sygnał zegarowy mógł ulec zmianie z 0 na 1. Ten minimalny czas liczony od momentu zmiany wartości wejścia D do momentu wystąpienia aktywnego (tutaj wznoszącego) zbocza zegara będziemy nazywać czasem wyprzedzenia (*set-up time*).

Pojęcie czasów utrzymania i wyprzedzenia dotyczy także innych przerzutników oraz układów synchronicznych, dlatego wielkości tych czasów są podawane w katalogach (dla 74LS74 czas utrzymania i wyprzedzenia wynosi odpowiednio 5ns i 20ns. Proponujemy Czytelnikowi poszukać w katalogach wartości tych czasów dla 74LS75).

16.2 Opis przerzutników

Wymienione dotychczas przerzutniki synchroniczne możemy opisywać za pomocą tabeli prawdy, bądź wyrażeń boolowskich. Dla przykładu przerzutnik JK można opisać za pomocą tabeli prawdy przedstawionej na rys.16.8a.

a)

y^t	J^t	K^t	y^{t+1}
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

b)

$J^t K^t$	00	01	11	10
$y^t = 0$			1	1
$y^t = 1$	1			1

$y^{t+1} = J^t \cdot \bar{y}^t + \bar{K}^t \cdot y^t$

Rys. 16.8 Przerzutnika JK: a - tablica prawdy, b - wyrażenie boolowskie

Oznaczenie y^t określa stan przerzutnika przed podaniem sygnału taktującego, a y^{t+1} stan przerzutnika osiągany po podaniu sygnału taktującego. W syntezie układów synchronicznych szczególnie przydatne są innego rodzaju opisy przerzutników. Otóż syntezując układ synchroniczny musimy wiedzieć, jakie wartości sygnałów należy podać na jego wejścia informacyjne, aby przerzutnik po podaniu sygnału taktującego zmienił stan y^t na y^{t+1} . Pomocne w tym celu są tablice wzbudzeń przerzutników. Z tego też względu podamy jedynie tablice wzbudzeń wymienionych wyżej typów

przerzutników (rys. 16.9).

$y^t \rightarrow y^{t+1}$	$S^t R^t$	$y^t \rightarrow y^{t+1}$	$J^t K^t$	$y^t \rightarrow y^{t+1}$	T^t	$y^t \rightarrow y^{t+1}$	D^t
0 \rightarrow 0	0 —	0 \rightarrow 0	0 —	0 \rightarrow 0	0	0 \rightarrow 0	0
0 \rightarrow 1	1 0	0 \rightarrow 1	1 —	0 \rightarrow 1	1	0 \rightarrow 1	1
1 \rightarrow 0	0 1	1 \rightarrow 0	— 1	1 \rightarrow 0	1	1 \rightarrow 0	0
1 \rightarrow 1	— 0	1 \rightarrow 1	— 0	1 \rightarrow 1	0	1 \rightarrow 1	1

przerzutnik SR, — $\in \{0,1\}$

przerzutnik JK, — $\in \{0,1\}$

przerzutnik T

przerzutnik D

Rys. 16.9 Tabele wzbudzeń podstawowych przerzutników

Przerzutnik typu D po podaniu sygnału taktującego pamięta to, co było na wejściu D przed podaniem tego sygnału, czyli opóźnia informację wejściową o jeden takt.

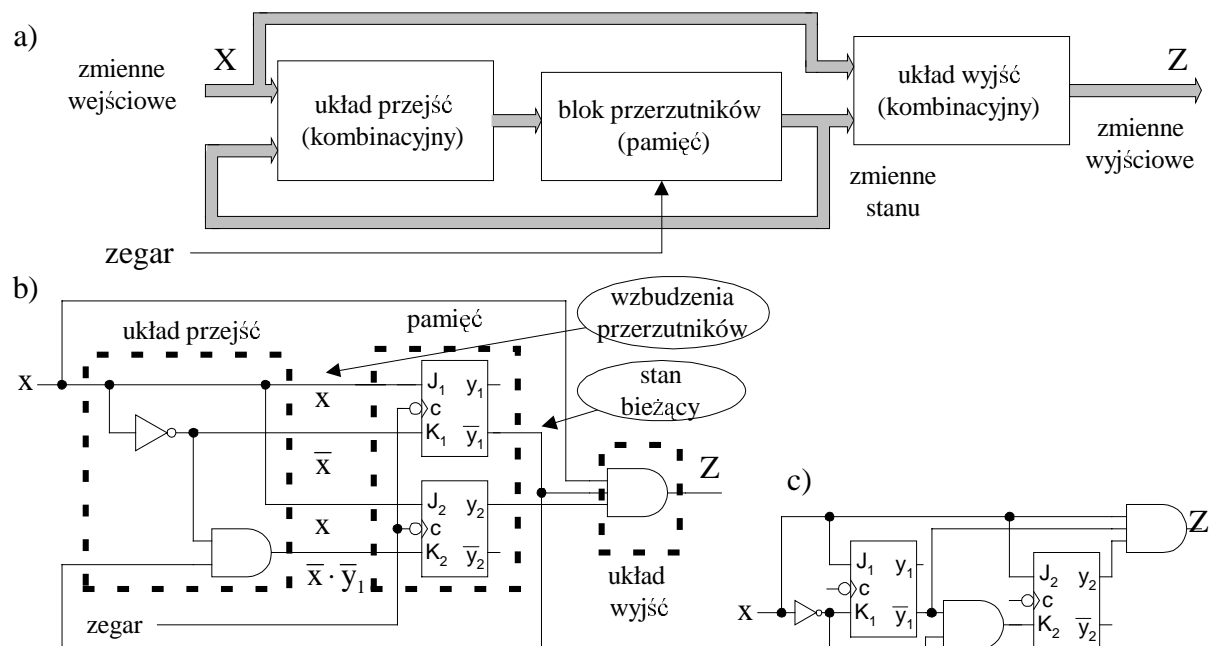
Podanie na wejście informacyjne przerzutnika T wartości 1 powoduje zmianę stanu przerzutnika na przeciwny po podaniu sygnału taktującego. Wartość 0 na wejściu T nie powoduje

zmiany jego stanu.

W przerzutniku JK, w odróżnieniu od przerzutników SR, są dozwolone wszystkie zestawy wartości sygnałów wejściowych.

16.3 Analiza układu synchronicznego

Ze względów dydaktycznych poprzedzenie syntezy wykładem z analizy układu jest celowe. Analizę rozpoczyna się od konkretnego schematu logicznego układu, którego funkcję należy odczytać, przypominamy sobie przy okazji takie pojęcia, jak stan, kodowanie stanów, tablica przejść-wyjść, aby można było użyć ich w bardziej świadomy sposób na etapie syntezy układu.



Rys. 16.10 Przykładowy układ synchroniczny: a - blokowa struktura, b - narysowany zgodnie ze strukturą blokową, c - narysowany w sposób "katalogowy"

Przyjmijmy zatem, że przez analizę układu synchronicznego będziemy rozumieć proces znajdowania dla danego schematu logicznego odpowiadającej mu tablicy przejść-wyjść, opisującej zachowanie układu synchronicznego.

Rozważmy układ synchroniczny, którego schemat logiczny przedstawiliśmy na rys.16.10. Określmy najpierw funkcje wzbudzeń przerzutników, czyli funkcje realizowane przez układy kombinacyjne, których wyjścia doprowadzone są do wejść przerzutników. Wartości funkcji wzbudzeń implikują wartości na wyjściach przerzutników po podaniu sygnału taktującego. Zauważmy, że w ogólnym przypadku funkcje wzbudzeń zależą od wartości sygnałów wejściowych, a poprzez pętlę sprzężenia zwrotnego od wartości przyjmowanych przez sygnały wyjściowe przerzutników. Funkcje wzbudzeń będziemy oznaczać literką, którą oznaczaliśmy wejście przerzutnika (indeks wskazuje na numer przerzutnika). Dla rozważanego układu z rys. 16.10 funkcje wzbudzeń są następujące:

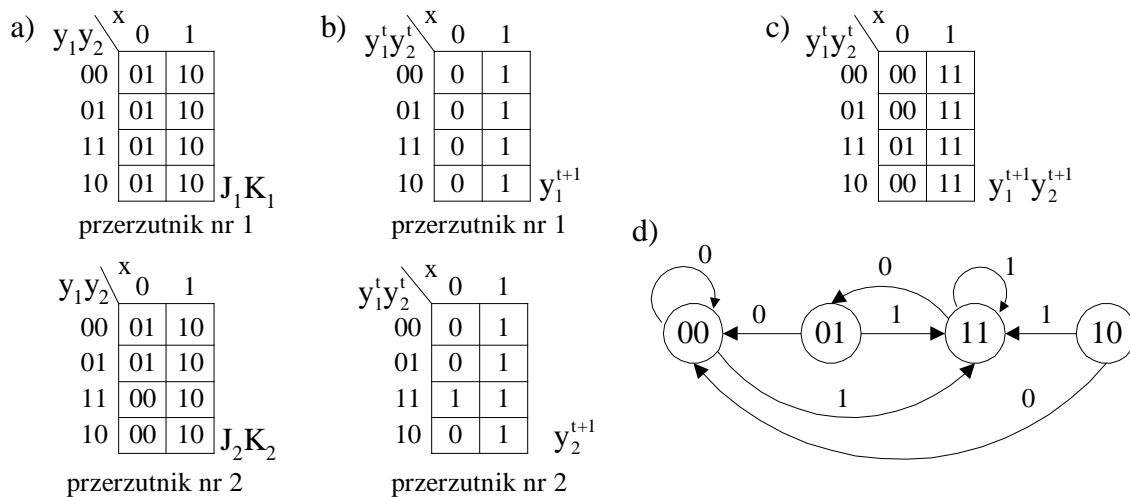
$$\text{przerzutnik nr 1 - } J_1 = x, \quad K_1 = \bar{x},$$

$$\text{przerzutnik nr 2 - } J_2 = x, \quad K_2 = \bar{x} \cdot \bar{y}_1,$$

Powyższe funkcje wzbudzeń wpiszemy do tablic Karnaugh. Dla każdego przerzutnika utworzymy osobną tablicę (rys. 16.11a), przy czym funkcje wzbudzeń wejść $J_i K_i$, $i=1,2$

wpiszemy do tej samej tablicy (dlatego w każdej kratce tablicy są dwie wartości).

Tablice funkcji wzbudzeń przedstawiają związek między wartościami przyjmowanymi przez wyjście y_i i-tego przerzutnika a wartościami wejść J_iK_i , zatem możemy określić na jego podstawie wartość sygnału wyjściowego przerzutnika po podaniu sygnału zegarowego. Przykładowo rozważmy kratkę w tablicy z rys.16.11a (przerzutnik nr 1) leżącą na przecięciu kolumny $x=0$ i wiersza $y_1^t y_2^t = 00$, do której wpisaliśmy wartości $J_1K_1=01$. Oznacza to, że przed podaniem wyróżnionej wartości sygnału zegarowego, $J_1=0$, $K_1=1$, a na wyjściu przerzutnika $y_1^t = 0$. Po podaniu wyróżnionej wartości sygnału zegarowego, zgodnie z tablicą wzbudzeń przerzutnika JK, sygnał wyjściowy przyjmie wartość $y_1^{t+1} = 0$ (nie uległ on zmianie). Gdy rozważymy kratkę leżącą na przecięciu $x=1$, $y_1^t y_2^t = 01$ (też dla przerzutnika nr 1), to po podaniu wyróżnionej wartości sygnału zegarowego otrzymamy $y_1^{t+1} = 1$. Przeprowadzając identyczne rozważanie dla wszystkich pozostałych kratek tablicy funkcji wzbudzeń układu, otrzymamy tablicę przejść (rys.16.11b) dla każdego przerzutnika z osobna.



Rys. 16.11 a - tablice funkcji wzbudzeń, b - tablice przejść przerzutników, c - tablica przejść układu, d - graf przejść układu

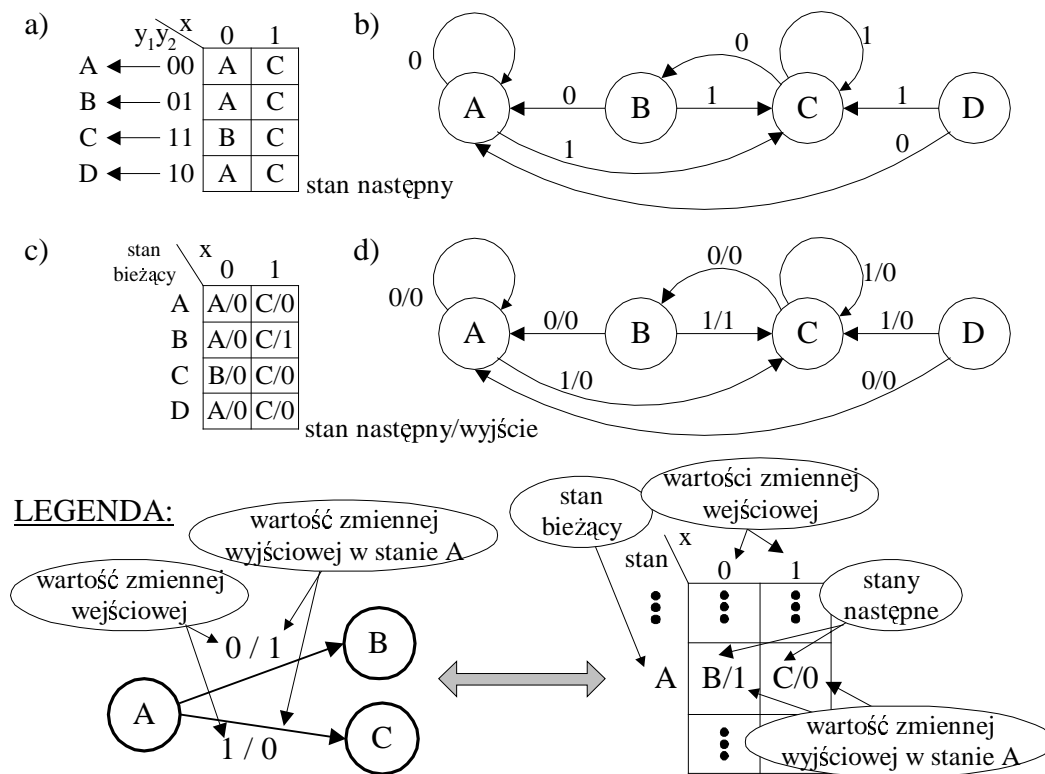
Naszym celem jest analiza całego układu, a nie tylko pojedynczych przerzutników, dlatego z obu tablic przejść (dla przerzutnika nr 1 i 2) utworzymy jedną tablicę przejść opisującą cały układ (rys.16.11c). Nazwa tablicy wywodzi się z tego, że opisuje ona przejścia z bieżącego zakodowanego stanu, reprezentowanego przez wartość $y_1^t y_2^t$, do stanu następnego, reprezentowanego przez wartości $y_1^{t+1} y_2^{t+1}$, pod wpływem wartości sygnałów wejściowych x . Tablicę przejść (rys. 16.11c) można przedstawić za pomocą grafu przejść (rys. 16.11d). Nad łukami skierowanymi grafu napisaliśmy wartości sygnału wejściowego, pod wpływem którego układ przechodzi od stanu bieżącego do stanu następnego, wskazywanego przez grot łuku skierowanego.

Dotychczasowe rozważania dotyczą pewnego konkretnego układu synchronicznego, dlatego każdy stan reprezentowany jest przez ściśle określony zestaw wartości zmiennych stanu y_1 i y_2 . Gdy zestawy te oznaczmy symbolicznie, np. stan $y_1 y_2 = 00$ nazwiemy A, stan $y_1 y_2 = 01$ nazwiemy B, itd., to otrzymamy tablicę stanów przedstawioną na rys. 16.12a i odpowiadający jej graf stanów na rys. 16.12b.

Każdy układ synchroniczny może być zrealizowany bądź w układzie Mealy'ego, bądź w układzie Moore'a. Przypomnijmy, że układem Mealy'ego nazywamy układ, którego funkcja wyjść ma postać $Z_i = f_j(x_1 \dots x_k y_1 \dots y_m)$ gdzie x_i - zmienne wejściowe, $i=1, \dots, k$, y_i - zmienne

stanu, $i=1, \dots, m$. Innymi słowy, aktualny stan wyjść zależy od aktualnego stanu wejść oraz aktualnego stanu wewnętrznego.

W układzie Moore'a aktualny stan wyjść zależy tylko od aktualnego stanu wewnętrznego. Wynika z tego, że aby rozpoznać rodzaj układu, wystarczy określić funkcje wyjść. W rozważanym przez nas układzie z rys.16.10a funkcja wyjścia $Z = x_1 \bar{y}_1 y_2$, czyli układ ten jest układem Mealy'ego. Z tego też względu funkcję wyjść dla układów Mealy'ego nanosi się na tablicę stanów, tworząc tablicę stanów-wyjść, ponieważ tablica stanów to nic innego jak kolejny sposób przedstawienia zależności między zmiennymi wejściowymi a zmiennymi stanu. Dlatego obok stanu następnego, do którego przechodzimy, wpisujemy po kresce ukośnej wartości zmiennych wyjściowych. W grafie stanów jest inaczej: wartości przyjmowane przez zmienne wyjściowe dopisujemy po kresce ukośnej obok wartości przyjmowanych przez zmienne wejściowe. Dla rozważanego układu tablice stanów i graf stanów z naniesionymi wartościami zmiennej wyjściowej z pokazaliśmy na rys.16.12c oraz 16.12d. Funkcje wyjść równie dobrze można nanieść na tablicę przejść (rys.16.11c).



Rys. 16.12 Tablice i grafy stanów

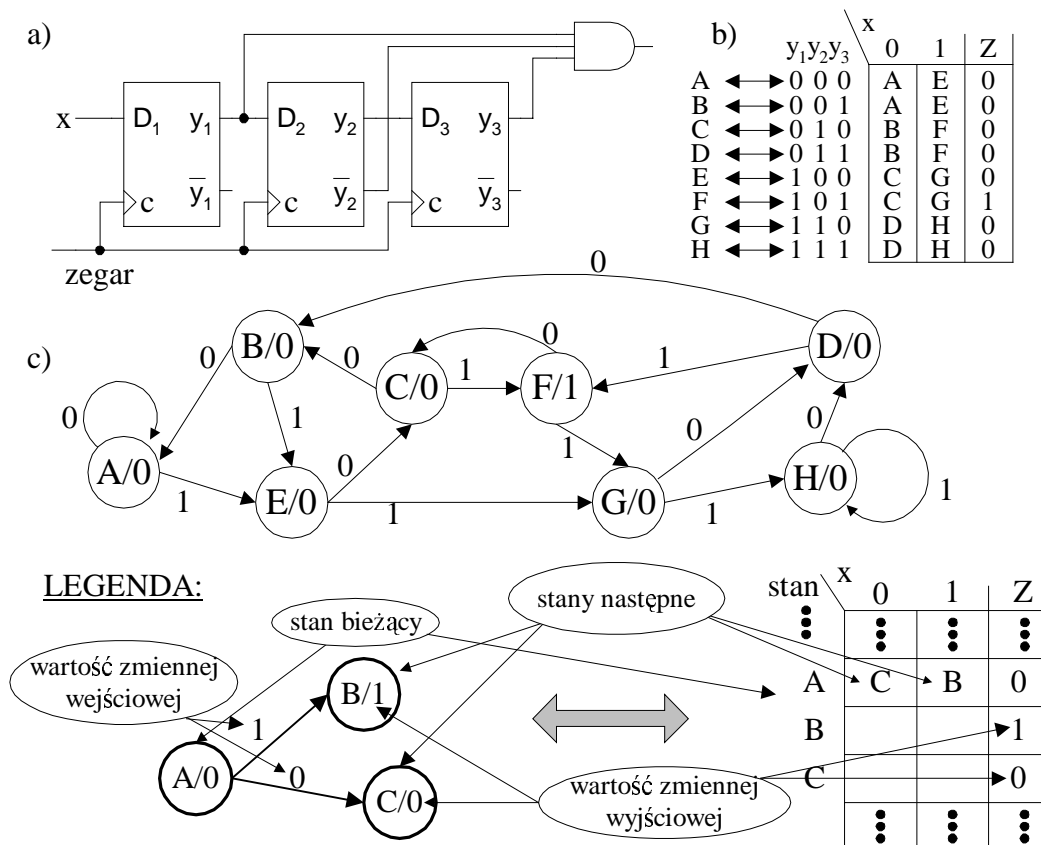
Uwaga: Przedstawiona konwencja dopisania wartości wyjść do tablicy stanów czy grafu stanów często prowadzi do nieporozumień. Powinniśmy pamiętać, że podana wartość wyjścia jest generowana cały czas, gdy znajdujemy się w pewnym stanie bieżącym i mamy ustalone wartości zmiennych wejściowych, a nie **podczas** przechodzenia do stanu następnego (jak często, mylnie się przyjmuje).

Śledząc graf stanów z rys.16.12d spróbujmy podać opis słowny działania układu. Dosyć łatwo zauważyć (?), że układ wykrywa sekwencje 101 podaną na jego wejście x , ponieważ zawsze $Z=1$, gdy kolejne trzy wartości zmiennej wejściowej $x^t=1$, $x^{t+1}=0$ i $x^{t+2}=1$. Każda inna sekwencja trzech kolejnych wartości zmiennej wejściowej powoduje, że wartość zmiennej wyjściowej $Z=0$.

Rozważmy z kolei układ z rys.16.13a. Składa się on z trzech szeregowo połączonych prze-

rzutników D oraz jednej bramki AND. Wiemy, że po podaniu sygnału zegara wartość sygnału podanego na wejście informacyjne przerzutnika D przeniesie się na jego wyjście. W omawianym układzie po podaniu sygnału zegara wartość sygnału x zostanie zapamiętana w przerzutniku D_1 , stan przerzutnika D_1 zostanie zapamiętany w przerzutniku D_2 , a wartość sygnału wyjściowego D_2 w przerzutniku D_3 . Innymi słowy, przerzutniki pamiętają kolejne trzy wartości z sekwencji wejściowej. Jeżeli założymy, że układ ma wykrywać sekwencję 101, wystarczy wykryć sytuację $y_1 y_2 y_3 = 101$, wykorzystując w tym celu bramkę AND realizującą funkcję $z = y_1 \bar{y}_2 y_3$. Tablice i graf stanów przedstawiliśmy na rys.16.13b i c.

Zatem układ z rys.16.13a wykrywa sekwencję 101, podobnie jak układ z rys.16.10, którego graf stanów rozważaliśmy.



Rys. 16.13 Układ Moore'a: a - schemat logiczny, b - tablica stanów, c - graf stanów

Zwróćmy uwagę na funkcję wyjść obu układów. Układ z rys.16.10 jest układem Mealy'ego, ponieważ $Z(x y_1 y_2) = x \bar{y}_1 y_2$ (argumentami są zmienne wejściowe i zmienne stanu), zaś układ z rys. 16.13a jest układem Moore'a, ponieważ funkcja wyjścia $Z(y_1 y_2 y_3) = y_1 \bar{y}_2 y_3$. Funkcja wyjścia dla układu Mealy'ego zależy od zmiennej wejściowej, dlatego jej wartość może ulegać zmianie tyle razy na jeden takt, ile razy zmienimy wartość x (np. przy $y_1 y_2 = 10$), podczas gdy w układzie Moore'a wartość funkcji wyjścia jest ściśle przypisana stanowi wewnętrznemu układu. W związku z powyższym, przy uruchamianiu układów Mealy'ego trzeba umieć właściwie interpretować reakcję układu na taktowanie impulsami zegarowymi i zmianę sygnałów wejściowych. Dlatego wydaje się, że układy Moore'a są łatwiejsze w uruchamianiu.

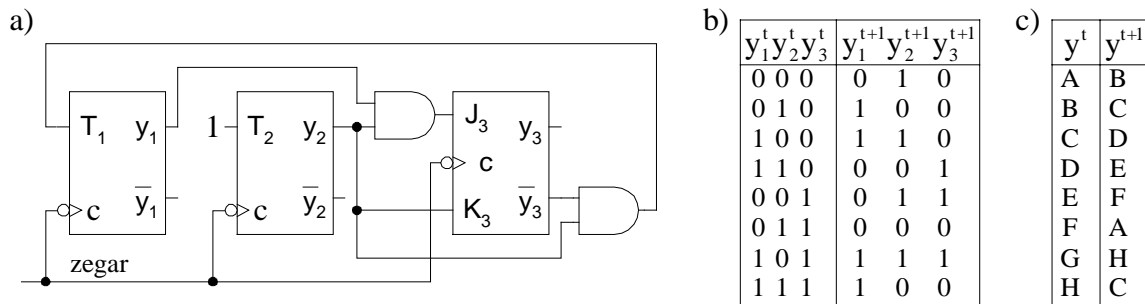
Przykład:

Przeanalizujemy układ synchroniczny z rys.16.14. Należy zwrócić uwagę, że układ wyko-

rzystuje różne rodzaje przerzutników: dwa przerzutniki T oraz jeden przerzutnik JK (czy wejścia zegarowe są dobrze oznaczone?). Układ nie ma wejść informacyjnych, jest zatem układem autonomicznym. Wypiszmy funkcje wzbudzeń przerzutników:

$$T_1 = y_2 \bar{y}_3, \quad T_2 = 1, \quad J_3 = y_1 y_2, \quad K_3 = y_2.$$

Zauważmy, że wszystkie funkcje wzbudzeń zależne są jedynie od zmiennych stanu układu.



Rys. 16.14 Układ autonomiczny: a - układ, b - tablica przejść, c - tablica stanów

Załóżmy, że układ znajduje się w stanie początkowym $y_1^t y_2^t y_3^t = 000$. Po podaniu pierwszego impulsu taktującego każdy z przerzutników zareaguje zgodnie z odpowiadającą mu funkcją wzbudzeń. Przerzutnik T_1 zmieni swój stan na przeciwny gdy $T_1=1$, dlatego przerzutnik nr 1 nie zmieni swojego stanu ($y_1^{t+1}=0$, ponieważ $T_1^t = y_2^t \bar{y}_3^t = 0 \cdot 1 = 0$). Drugi przerzutnik T_2 będzie zmieniał swój stan na przeciwny za każdym razem po podaniu impulsu taktującego, ponieważ na jego wejście informacyjne podajemy stałą wartość 1. Trzeci przerzutnik nie zmieni swojego stanu (będzie $y_3^{t+1}=0$), ponieważ $y_3^t=0$ oraz $J_3^t K_3^t = 00$. Oznacza to, że ze stanu $y_1^t y_2^t y_3^t = 000$ układ przejdzie do stanu $y_1^{t+1} y_2^{t+1} y_3^{t+1} = 010$. Powtarzając rozważanie przekonamy się, że po podaniu drugiego impulsu taktującego układ przejdzie do stanu $y_1^{t+1} y_2^{t+1} y_3^{t+1} = 100$, ponieważ $T_1^t = 1, T_2^t = 1$ oraz $J_3^t K_3^t = 01$. Kontynuując powyższe rozważania dojdziemy do tablicy przejść pokazanej na rys. 16.14b.

W układach autonomicznych przejście ze stanu bieżącego (Y^t) do stanu następnego (Y^{t+1}) odbywa się tylko pod wpływem sygnału taktującego, dlatego w grafie stanów nad łukami łączącymi stany nie piszemy wartości zmiennych wejściowych.

16.4 Opis układu synchronicznego

Opis zachowania się układu synchronicznego, czyli przedstawienie zależności między wartościami wyjściowymi a wejściowymi układu, można wyspecyfikować na kilka sposobów. Naturalnym opisem jest opis słowny, który z wiadomych przyczyn może wносить dwuznaczności interpretacyjne. Opis słowny wymaga starannego oraz precyzyjnego formułowania myśli, co z kolei z dydaktycznego punktu widzenia jest jego cenną zaletą. Do opisu układu możemy wykorzystać diagram czasowy. Na jednej osi czasu zaznaczamy zmiany wartości sygnałów wejściowych, a na drugiej, umieszczonej pod pierwszą, nanosimy odpowiednie zmiany sygnałów wyjściowych.

Zachowanie się układu synchronicznego możemy również opisać za pomocą tablicy przejść wyjść. W praktyce rzadko się zdarza, że w chwili przystąpienia do syntezy mamy od razu podaną tablicę przejść-wyjść. Wychodząc od diagramu czasowego lub opisu słownego staramy się otrzymać tablicę przejść-wyjść, która jednoznacznie opisuje zachowanie się

układu oraz jest punktem wyjścia dla metody syntezy. Tablice przejść-wyjść można otrzymać przyjmując za punkt wyjścia takie opisy, jak wyrażenia regularne czy sieci działań.

16.5 Synteza układu synchronicznego

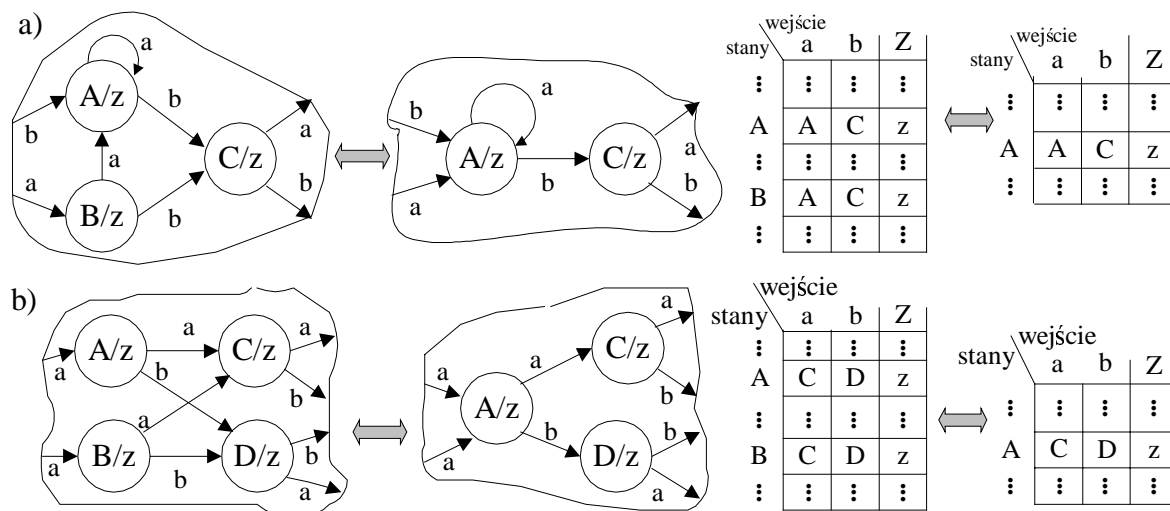
Proces syntezy układu synchronicznego przebiega w kierunku odwrotnym w stosunku do analizy przedstawionej w poprzednich rozdziałach. Przyjmujemy, że punktem wyjścia do syntezy jest tablica przejść-wyjść (lub tablica stanów z naniesionymi wartościami wyjść). Tablicę tę otrzymujemy wychodząc od takiego dowolnego opisu zachowania się układu, który najlepiej odzwierciedla myśli konstruktora (np. opis słowny, diagram czasowy itp.).

Charakterystyczne etapy syntezy możemy przedstawić następująco:

1. Specyfikacja problemu (opis słowny, wykres czasowy, graf stanów),
2. Utworzenie tablicy stanów i przejść (z ewentualną zmianą układu Mealy'ego w Moore'a lub odwrotnie),
3. Minimalizacja liczby wierszy tablicy stanów-wyjść,
4. Kodowanie tablicy stanów,
5. Określenie funkcji wzbudzeń przerzutników i funkcji wyjść,
6. Narysowanie układu.

Omówimy je teraz, zwracając uwagę na specyficzne aspekty przynależne do każdego etapu, najpierw jednak przybliżymy każdy z nich.

Z opisu problemu otrzymujemy tablice stanów i tablice wyjść (np. takie jak na rys. 16.12). Tablice te jednoznacznie opisują zachowanie się układu. Ponieważ proces przechodzenia od opisu (specyfikacji) do tablicy stanów-wyjść może być różny (zależny nie tylko od użytego formalizmu), może się zdarzyć, że dwie osoby uzyskają różne tablice stanów-wyjść opisujące zachowanie się tego samego układu. Oznacza to, że układy uzyskane z obu tablic stanów-wyjść będą jednakowo reagować na podawane sekwencje sygnałów wejściowych. Oczywiście, oba układy będą prawdopodobnie różnić się liczbą użytych elementów logicznych (bramki, przerzutniki).

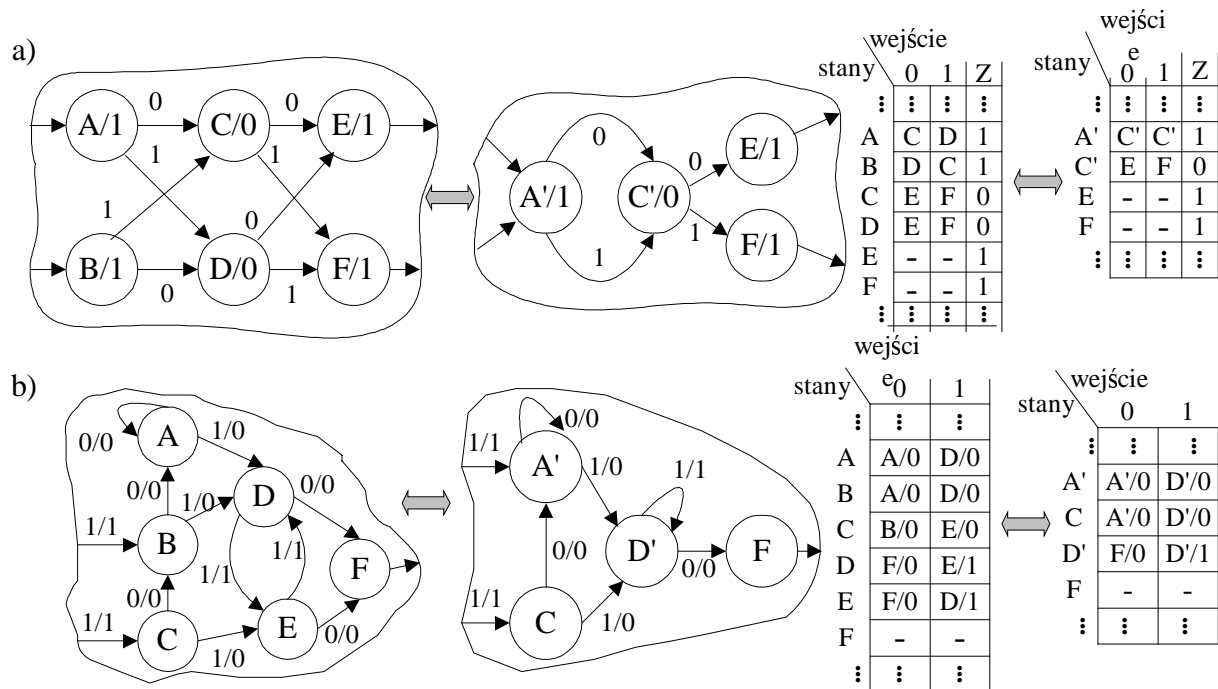


Rys. 16.15 Równoważność stanów

Zastanówmy się, z czego wynika fakt identycznego zachowania się układów, pomimo że opisane są za pomocą różnych tablic stanów-wyjść.

W tym celu rozważmy przykładowe podgrafy, stanowiące części pewnych większych grafów stanów (rys. 16.15). Obok podaliśmy równoważne im podgrafy wraz z częścią tablicy stanów-wyjść, odpowiadającą danym podgrafom. Rozpatrujemy grafy odpowiadające układom typu Moore'a. Literą wejściową nazwiemy ustalony zestaw wartości podawanych na każde z wejść układu. Jeżeli układ jest dwuwejściowy, to możliwych liter wejściowych jest cztery, które możemy oznaczyć jako a, b, c i d. W rozważanych podgrafach występują tylko dwie litery wejściowe, a i b.

Weźmy pod uwagę podgraf z rysunku 16.15a, zwracając szczególną uwagę na stanach A i B, którym przypisano identyczne wartości sygnałów wyjściowych "z" oraz z obu stanów przechodzimy do stanu C pod wpływem litery b.



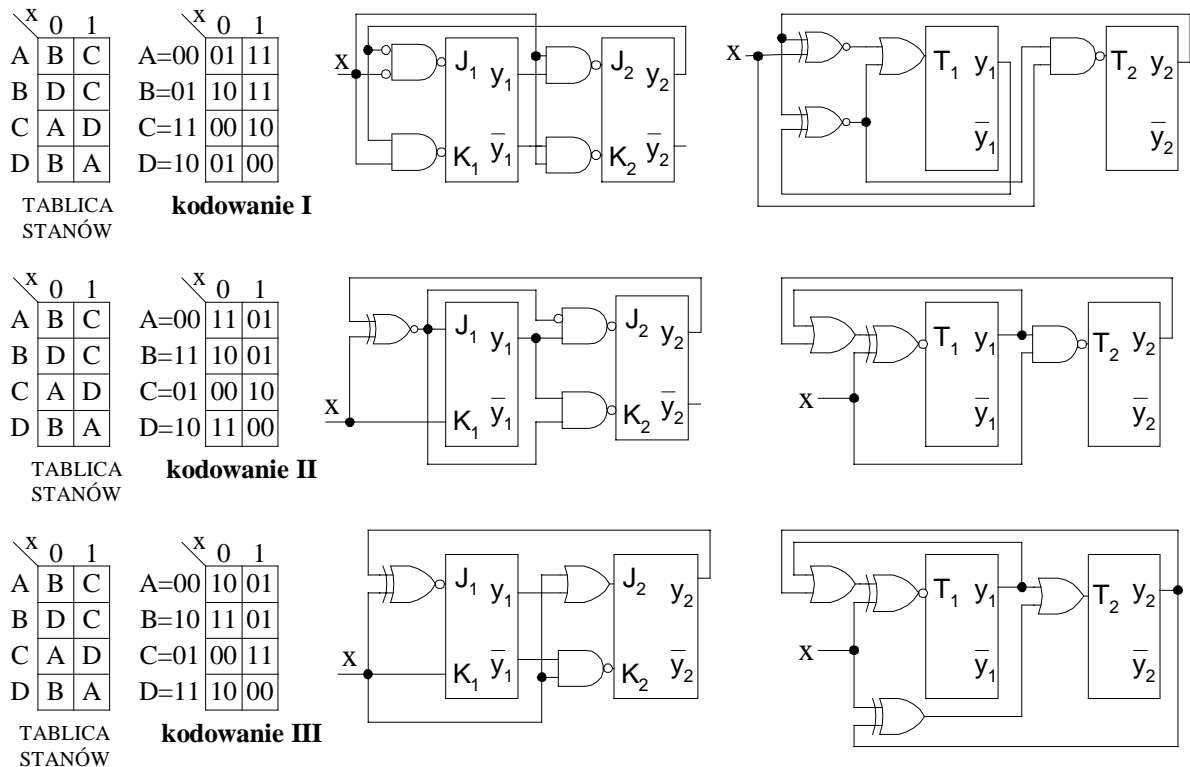
Rys. 16.16 Równoważność stanów

Zauważmy, że bez względu na to, czy układ jest w stanie A czy B, podanie na wejściu litery a spowoduje przejście do stanu A/z. Jeżeli będąc w rozważanych stanach podamy na wejście literę b - przejdziemy do stanu C/z. Znaczy to, że stany A i B są stanami równoważnymi, ponieważ bez względu na to, w którym z nich przebywa układ, podanie na wejście tych samych liter wejściowych daje identyczny skutek na wyjściu układu, tzn. po podaniu litery a na wyjściu otrzymamy literę z, natomiast gdy podamy b, na wyjściu otrzymamy z. Innymi słowy, jeden z tych stanów jest nadmiarowy i możemy się go pozbyć.

Na rysunku 16.15b przedstawiliśmy kolejny przykład równoważności pary stanów. Przeprowadzając rozumowanie analogiczne do powyższego dochodzimy do wniosku, że stanami równoważnymi są stany A/z oraz B/z.

Po każdorazowym wyeliminowaniu jednego z pary stanów równoważnych możemy sprawdzić, czy pozostawiony stan jest równoważny z innymi. Jeżeli dla żadnego stanu nie istnieją już stany z nim równoważne, proces poszukiwania stanów równoważnych należy uznać za zakończony. Innymi słowy, proces minimalizacji tablicy stanów-wyjść kończymy, jeżeli znajdziemy maksymalne zbiory par stanów równoważnych, czyli takie zbiory stanów, których liczność jest maksymalna, a wszystkie stany należące do tych zbiorów są parami równoważne. Oczywiście w zminimalizowanej tablicy pozostawiamy jedynie reprezentanta danego maksymalnego zbioru stanów równoważnych.

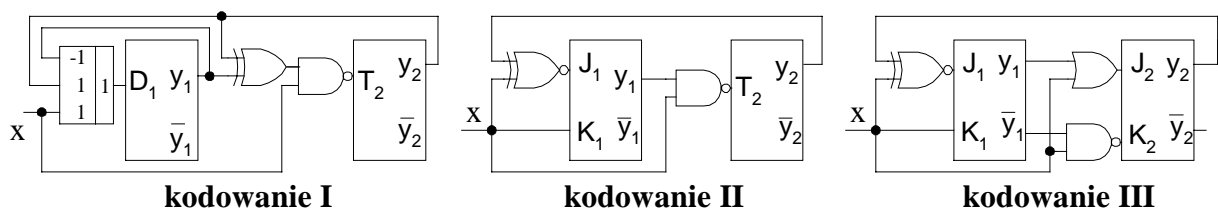
Proponuje się Czytelnikowi rozważenie kolejnych przykładów równoważnych grafów i tablic, które przedstawiliśmy na rys. 16.16.



Rys. 16.17 Kodowanie a złożoność układu - ilustracje (wejścia zegarowe pominięto)

Po otrzymaniu minimalnej tablicy stanów należy tablicę zakodować, tzn. wzajemnie jednoznacznie przypisać każdemu ze stanów wartości przyjmowane przez zmienne stanu (y_1, y_2, \dots, y_k). W odróżnieniu od układów asynchronicznych, w układach synchronicznych nie występuje zjawisko wyścigów, dlatego problem zakodowania układu wydaje się zabiegiem prostym.

Oczywiście jest to prawda, z tym tylko, że w zależności od doboru kodów dla tej samej tablicy stanów otrzymamy układy różniące się swoją złożonością (liczba bramek, połączeń międzyelementowych, ścieżek rozbieżnych itp.). Ilustruje to rysunek 16.17.



Rys. 16.18 Prostsze realizacje układowe tablicy stanów z rys. 16.17

Jeżeli zdecydujemy się na wybór kodu, to złożoność układu będzie zależała od wyboru rodzaju przerzutnika. Ilustruje to rysunek 16.18, na którym pokazaliśmy prostsze realizacje tablicy stanów z rysunku 16.17.

Po zakodowaniu tablicy stanów (otrzymujemy tablicę przejść) należy zdecydować o wyborze przerzutników, a następnie dla każdego z jego wejść, określić funkcje wzbudzeń. Funkcje wzbudzeń realizowane są przez układy kombinacyjne, których wyjścia podajemy na wejścia przerzutników. Funkcje te otrzymujemy z tablic przejść, ponieważ tablica ta to nic

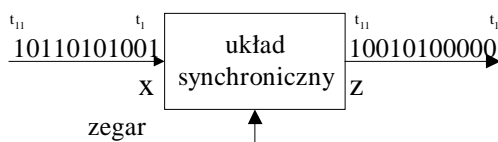
innego jak zależność pomiędzy aktualnymi wartościami przyjmowanymi przez każdą ze zmiennych stanu a wartościami, jakie każda ze zmiennych powinna przyjąć po podaniu impulsu zegarowego. Innymi słowy, przerzutnik, który reprezentuje daną zmienną stanu, powinien mieć podane na swoje wejścia informacyjne takie wartości (wzbudzenia), by wartość jego wyjścia (zmienna stanu) zmieniała się zgodnie z tablicą przejść. Jeżeli otrzymamy minimalne realizacje wszystkich funkcji wzbudzeń, możemy narysować schemat logiczny układu (jeszcze bez układu wyjść).

16.5.1 Tablica stanów-wyjść

Po wyjaśnieniu podstawowych zagadnień oraz problemów występujących podczas syntezy układów synchronicznych przejdźmy do poniższego przykładu, którego celem będzie skonstruowanie tablicy stanów-wyjść:

Przykład:

Zsyntezuj układ synchroniczny, który po podaniu na jego wejście sekwencji 101 generuje wartość sygnału wyjściowego $z=1$. W przypadku przeciwnym $z=0$.



Rys. 16.19 Ilustracja do przykładu

Z opisu słownego wynika, że układ ma jedno wejście, które oznaczymy przez x oraz jedno wyjście z . Zasadę działania układu ilustruje rysunek 16.19. Sekwencje zerojedynkową podawaną na wejście x należy rozumieć następująco: w chwili t_1 , $x=1$; t_2 , $x=0$; t_3 , $x=0$; t_4 , $x=1$; t_5 , $x=0$ itd. Podobnie sekwencja wyjściowa: w chwili t_1 , t_2 , t_3 , t_4 , t_5 , $z=0$;

t_6 , $z=1$; t_7 , $z=0$, itd. Poszczególne chwile wyznaczone są przez wyróżnione wartości sygnału zegarowego.

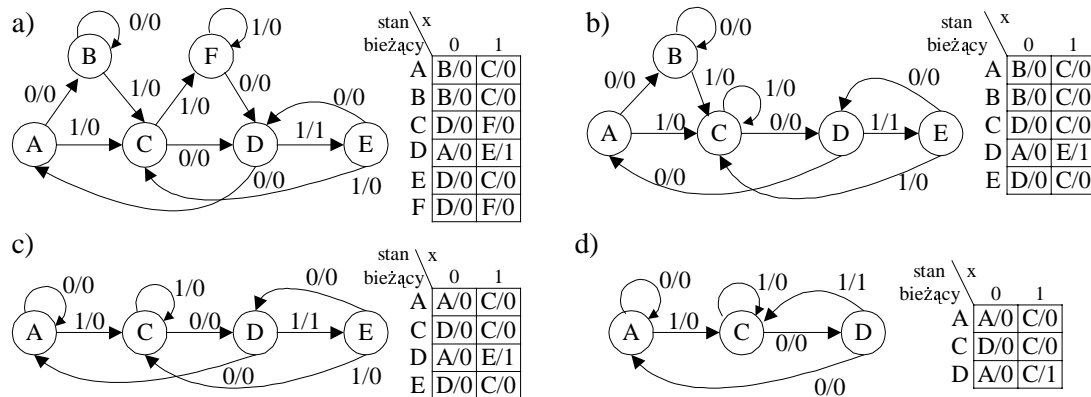
Rozważania nasze rozpoczniemy od utworzenia grafu stanów, ponieważ wydaje się, że łatwiej jest utworzyć graf niż tablicę stanów-wyjść. Dopiero w następnym kroku na podstawie grafu utworzymy tablicę stanów-wyjść.

Na wstępie założmy, że układ znajduje się w stanie początkowym A (o stanie początkowym powiemy więcej w następnych rozdziałach). Różne osoby, przystępujące do tworzenia grafu stanów, mogą otrzymać różne jego postacie. Ważne jest, aby każdy z tych grafów opisywał działanie układu zgodne ze specyfikacją (w tym przypadku specyfikacją słowną). Na czym zatem polega różnica między grafami? Wynika ona z faktu ignorowania nadmiarowych stanów równoważnych na etapie tworzenia grafu. Po minimalizacji liczby stanów musimy otrzymać ten sam graf (z dokładnością do oznaczeń).

Założmy, że powyższy przykład zaczęły syntezować cztery osoby, a wynikiem ich pracy są grafy, które przedstawiliśmy na rys. 16.20. Zastanówmy się, jakie rozumowanie mogła przeprowadzić osoba tworząca graf stanów z rys. 16.20c. Zapewne przyjęła, że stanem początkowym jest stan A. Od tego stanu rozpoczęło się tworzenie grafu. Następnie prawdopodobnie przyjęła, że kolejna poprawna wartość wejściowa, tzn. pochodząca z rozpoznawanej sekwencji wejściowej, przeprowadzi układ do nowego stanu. W ten sposób powstały stany C, D i E. Stan C reprezentuje następującą historię wejść - podano pierwszą jedynkę z rozpoznawanej sekwencji, stan D - podano jedynkę i zero, stan E - podano jedynkę, zero i jedynkę. Dochodząc do stanu E, układ rozpoznaje sekwencje 101, a więc zgodnie z opisem słownym $z=1$. Ponieważ na układ znajdujący się w dowolnym z wymienionych stanów możemy podać wartość $x=0$ albo $x=1$ należy zdecydować, do jakich stanów następnych należy przejść, gdy podamy na wejście wartości jeszcze nie brane pod uwagę.

Założmy więc, że układ znajduje się w stanie A. Zdecydowano już, że do stanu C układ przejdzie, gdy $x=1$. Należy zatem zdecydować, do którego stanu układ przejdzie, gdy $x=0$.

Zgodnie z rys. 16.20c układ ma pozostać w stanie A. Rozumowanie jest słuszne, ponieważ stan A jest stanem początkowym i pierwsze (oraz następne kolejne zera na wejściu) nie rozpoczynają rozpoznawanej sekwencji. Dopiero podanie pierwszej jedynki spowoduje przejście do stanu C, ponieważ jedynka ta być może stanowi początek rozpoznawanej sekwencji 101. W stanie C, gdy podamy na wejście układu zero, układ przejdzie do stanu D, ponieważ jest to druga poprawna wartość z sekwencji dlatego należy ten fakt zapamiętać. Układ jest w stanie „pierwsza jedynka z rozpoznawanej sekwencji” i podajemy na wejście jedynkę. Nie



Rys. 16.20 Różne grafy i tablice stanów-wyjść opisujące działanie tego samego układu

powinno się nic zmieniać, bo aktualnie podana jedynka i kolejne następne będą być może pierwszymi w rozpoznawanej sekwencji - dlatego układ pozostanie w stanie C.

Jeżeli układ znajduje się w stanie D, to po podaniu na wejście wartości zero przejdzie do stanu A. Rzeczywiście, ponieważ stan D odpowiada sytuacji częściowego rozpoznania dwóch pierwszych wartości sekwencji, zatem podanie na jego wejście zera spowoduje, że proces rozpoznawania powinien rozpocząć się od „początku”. Wynika to z tego, że po sekwencji 10 podanie na wejście zera nie rozpoczyna rozpoznawanej sekwencji. Stan E odpowiada sytuacji „sekwencja 101 została rozpoznana”, co sygnalizowane jest wartością wyjścia $z=1$. W stanie E po podaniu jedynki układ przechodzi do stanu C, ponieważ jest to druga kolejna jedynka w sekwencji (pierwszą podaliśmy dochodząc do stanu E), więc być może rozpoczyna kolejną sekwencję. Jeżeli w stanie E podamy na wejście wartość zero, to układ przejdzie do stanu D (dosyć często studenci decydują się na przejście do stanu A). Zauważmy, że po podaniu na wejście układu sekwencji 101 przejdzie on do stanu E. Ostatnią wartością podawaną na wejście była jedynka, i jeżeli jako kolejną wartość podajemy zero, układ powinien przejść do stanu częściowego rozpoznania sekwencji 10, czyli do stanu D. Przejście układu do stanu A odpowiadałoby zupełnie innemu opisowi działania układu (podaj ten opis). Na rys 16.20 obok każdego grafu stanów, podaliśmy odpowiadającą mu tablicę stanów-wyjść (układ Mealy’ego).

16.5.2 Minimalizacja liczby stanów

Istotę minimalizacji liczby stanów wewnętrznych układu synchronicznego już poznaliśmy, rozważając grafy z rys. 16.15 i 16.16. Ponieważ istnieją zalgorytmizowane metody minimalizacji, a omawiając je używa się uznanych pojęć, więc należy się z tymi pojęciami zapoznać.

Dodatkowo, metody i pojęcia używane przy minimalizacji liczby stanów są zróżnicowane w zależności od tego, czy rozważamy układ zupełny czy też niezupełny. Niezupełnym układem synchronicznym nazywamy układ, w którym nie określono wszystkich stanów następnych bądź nie określono stanu wyjścia. Na tablicy stanów-wyjść odpowiada to sytuacji

nieokreślenia wewnątrz tablicy stanu-wyjścia (zaznaczamy ten brak za pomocą kreski w miejscu stanu lub wyjścia). Układami niepełnymi nie będziemy się zajmować, tzn. wszystkie pojęcia i metody podane dalej będą słuszne dla układów pełnych. Wiemy już, że celem minimalizacji liczby stanów (wierszy tablicy stanów-wyjść) jest znalezienie dla tej tablicy równoważnej jej tablicy stanów-wyjść mającej minimalną liczbę wierszy.

Dwa układy uznamy za równoważne, jeżeli dla dowolnej sekwencji wejściowej na ich wyjściach otrzymujemy identyczne sekwencje wyjściowe. Mówiąc inaczej, jeżeli układy potraktujemy jako „czarne skrzynki”, których struktury są różne, a obserwować możemy jedynie stany wejść i wyjść, to nie będziemy mogli tych układów rozróżnić.

Definicja równoważności może być sformułowana poprzez równoważność pary stanów.

Definicja 16.1

Dwa stany A, B w tablicy stanów-wyjść nazywamy równoważnymi w synchronicznym układzie pełnym, gdy:

1. wartości zmiennych wyjściowych w obu stanach są jednakowe dla wszystkich możliwych sygnałów wejściowych,
2. stany, do których układ przechodzi ze stanów A i B, są jednakowe lub równoważne dla wszystkich możliwych sygnałów wejściowych.

Zauważmy, że definicja ma charakter rekurencyjny, tzn. dwa stany mogą być równoważne, jeżeli para innych stanów jest równoważna. Mówimy wtedy o równoważności warunkowej.

W pierwszym kroku proces minimalizacji liczby stanów będzie polegał na znalezieniu wszystkich par stanów równoważnych. Spośród wszystkich par należy wybrać te pary, które tworzą zbiór stanów równoważnych. Dla przykładu, jeżeli $A \equiv B$, $A \equiv C$ oraz $B \equiv C$, gdzie \equiv jest oznaczeniem równoważności pary stanów, to ze względu na przechodność relacji równoważności rozważane trzy stany możemy zastąpić jednym stanem. Najlepiej, gdy zbiór stanów równoważnych będzie maksymalny, tzn. będzie zawierał możliwie największą liczbę wzajemnie równoważnych par stanów.

Głównym celem procesu minimalizacji jest zatem znalezienie rodziny maksymalnych zbiorów stanów równoważnych. W tym celu posłużymy się tablicą trójkątną, z którą zapoznaliśmy się przy poszukiwaniu stanów pseudorównoważnych w tablicy Huffmana. Na wstępie rysujemy tablicę trójkątną (rys 16.21), której współrzędne odpowiadają wartościom stanów wewnętrznych w tablicy stanów.

Kratkę o współrzędnych (i,j) wykreślamy, jeżeli stany A_i i A_j układu mają sprzeczne (różne) wyjścia. Gdy równoważność pary stanów A_i i A_j jest uwarunkowana równoważnością pewnej pary stanów C_k i C_l , wówczas parę k, l wpisujemy do kratki o współrzędnych (i,j) (oczywiście, jeżeli para stanów i, j jest uwarunkowana równoważnością pary i, j, to nie ma potrzeby wpisywania tej pary do kratki (i,j)). Następnie wykreślamy wszystkie kratki, w których jako warunek (lub jeden z warunków), wpisana jest para k, l odpowiadająca wykreślonej kratce (k,l) na poprzednim etapie. Czynność wykreślania kratek prowadzi się aż do uzyskania sytuacji, gdy wszystkie pary określające warunki odpowiadają niewykreślonym kratkom.

Na rysunku 16.21 pokazaliśmy wypełnione tablice trójkątne dla odpowiednich tablic stanów-wyjść z rys. 16.20a, b, c, d. W tak uzyskanych tablicach wszystkie nie skreślone kratki, bez względu na ich zawartość, odpowiadają parom stanów równoważnych.

Na podstawie tablicy trójkątnej możemy uzyskać graf równoważności stanów. W grafie tym zaznaczamy wszystkie stany parami równoważne. Wystarczy teraz połączyć w grupy te stany, które w grafie są połączone **każdy z każdym**, by otrzymać rodziny maksymalnych

zbiorów stanów równoważnych. Grafy równoważności przedstawiliśmy na rys. 16.21 wraz z maksymalnymi zbiorami stanów równoważnych.

Rodzinę maksymalnych zbiorów stanów równoważnych można też wyznaczyć posługując się wyrażeniami boolowskimi. Mianowicie, w celu otrzymania maksymalnego zbioru stanów równoważnych należy ze zbioru wszystkich stanów usunąć po jednym stanie z pary stanów nierównoważnych (wykreślonych z tablicy trójkątnej). Dla tablicy z rysunku 16.21c należy usunąć stan A lub C, A lub D, A lub E, C lub D oraz D lub E.

Powyższe stwierdzenia możemy zapisać za pomocą wyrażenia boolowskiego

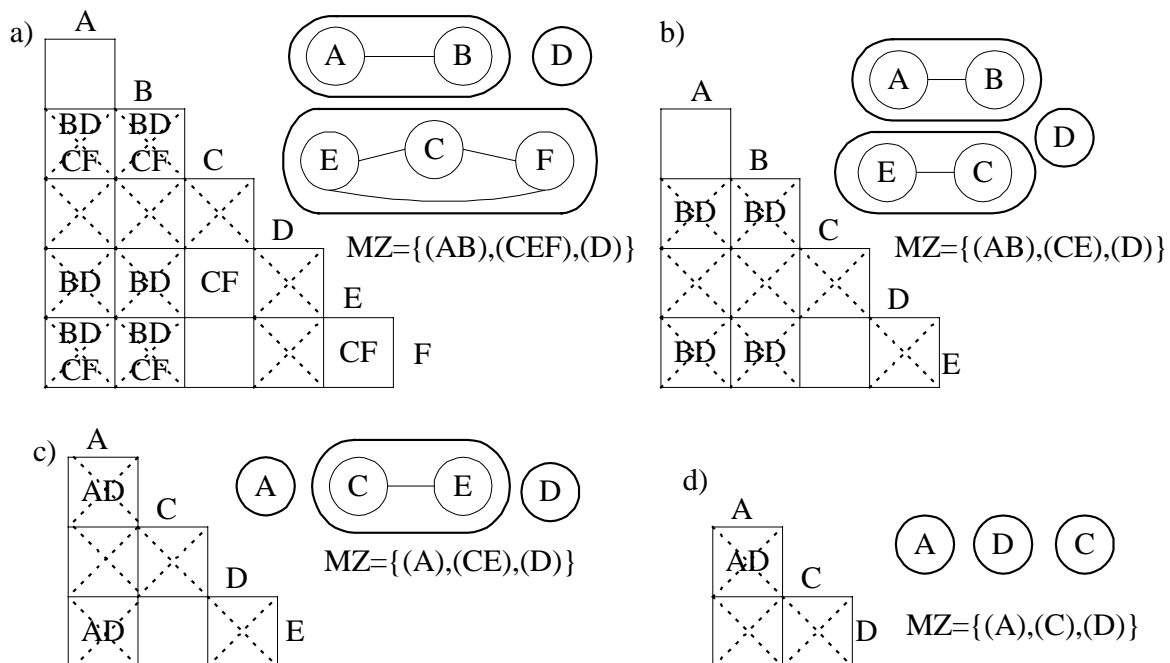
$$(A+C)(A+D)(A+E)(C+D)(D+E)=1$$

Grupując odpowiednio czynniki wyrażenia oraz stosując prawo $(a+b)(a+c)=a+bc$, otrzymamy:

$$(A+CDE)(D+CE)=1,$$

$$AD+ACE+CDE=1.$$

Interpretacja tego zapisu jest następująca: ze zbioru usunąć stany A i D lub A i C oraz E lub C i D oraz E. Pozostawione stany będą tworzyć maksymalne zbiory stanów równoważnych. Dla powyższego przykładu będą nimi (CE), (D), (A).



Rys. 16.21 Tablice trójkątne oraz grafy równoważności dla tablic stanów z rys. 16.20

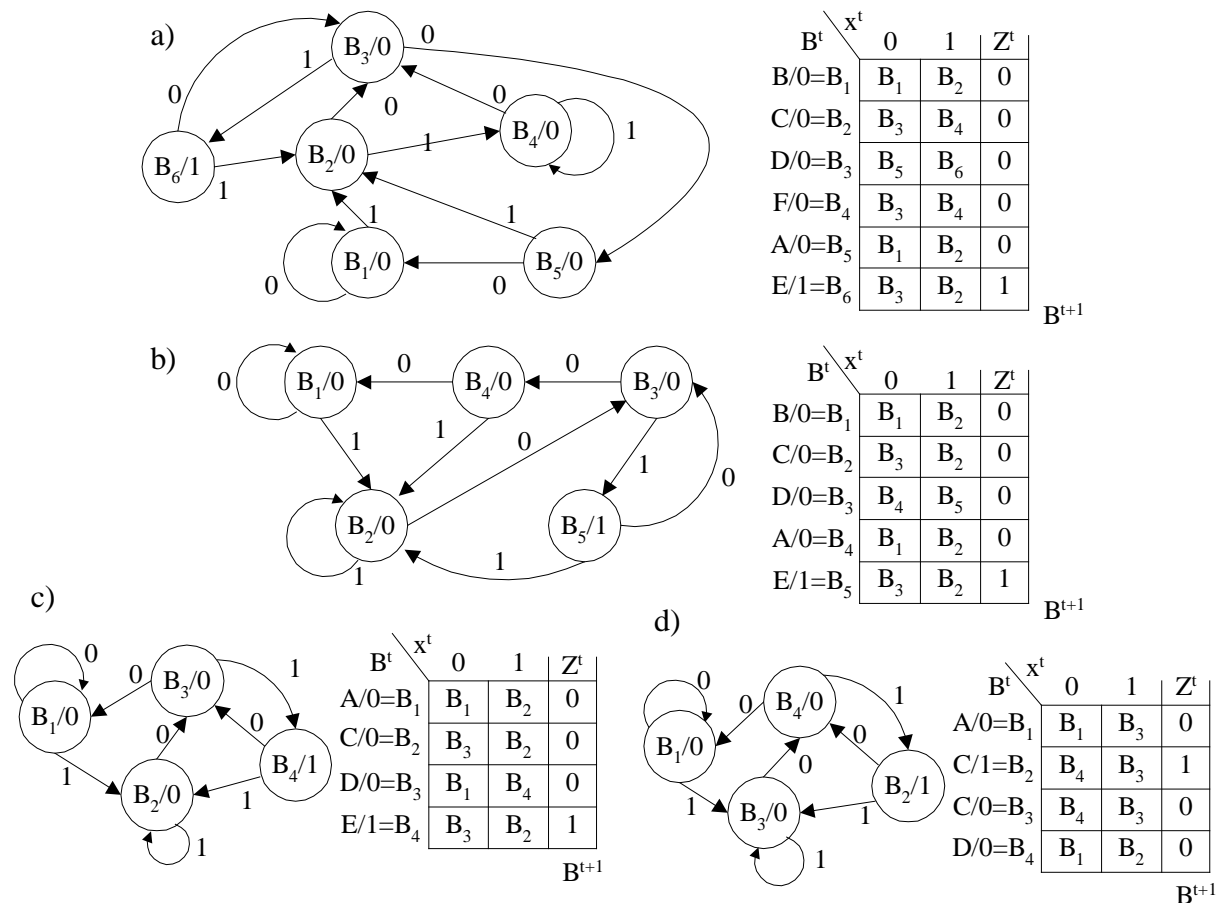
Pierwsze podejście, wykorzystujące graf równoważności, stosujemy wówczas, gdy w tablicy trójkątnej jest mniej kratek nie wykreślonych, podejście drugie - gdy mniej jest kratek wykreślonych. Dla rozważanego przykładu minimalną tablicą stanów-wyjść jest tablica z rys. 16.20d.

16.5.3 Zmiana układu Mealy'ego na Moore'a i odwrotnie

Złożoność układu synchronicznego zależy między innymi od wyboru typu układu. Zazwyczaj układy Mealy'ego są mniej złożone niż układy Moore'a. Z drugiej strony, jak już wspo-

mnieliśmy, układy Moore'a są łatwiejsze w uruchamianiu. Dlatego dobrze jest zsyntezować układy obu rodzajów i wybrać korzystniejszy, ze względu na przyjęte kryterium wyboru.

Wychodząc od opisu słownego (lub innej specyfikacji), otrzymujemy zazwyczaj tablice stanów-wyjść jednego rodzaju. Należy więc znać sposób przekształcenia układu Mealy'ego w układ Moore'a i odwrotnie.



Rys. 16.22 Tablice i grafy układu Moore'a odpowiadające tablicom i grafom układu Mealy'ego z rys. 16.20

Tablicę stanów-wyjść układu Mealy'ego w tablicę stanów-wyjść równoważnego mu układu Moore'a można zmienić w następujący sposób:

1. Każdej różnej parze (stan, wyjście), np. A_i/Z_k , wewnątrz tablicy stanów-wyjść odpowiadającej układowi Mealy'ego przyporządkowujemy dowolny symbol, np. B_j .
2. Symbole B_j odpowiadają stanom układu Moore'a. Każdemu B_j przypisujemy stan wyjść Z_k . Stany następne dla każdego stanu X_i są takie same, jakie miał odpowiadający symbolowi B_j stan A_i układu Mealy'ego. Jeżeli w tablicy Mealy'ego istnieje stan, który nigdy nie jest stanem następnym (dlatego nazywany stanem startowym), to należy mu przyporządkować symbol B_j z nieokreślonym stanem wyjść albo arbitralnie przyjąć pewną wartość.

Przejście z układu Moore'a na układ Mealy'ego dokonujemy następująco:

1. Wewnątrz tablicy stanów-wyjść układu Moore'a, obok stanów następnych B_i , wpisujemy wartości wyjścia układu Moore'a odpowiadające temu stanowi.
2. Usuwamy kolumny wartości wyjściowych i uzyskaną w punkcie 1 tablicę stanów-wyjść minimalizujemy.

Jako przykład rozważmy tablicę układu Mealy'ego z rys 16.20d. Zgodnie z algorytmem przejścia z układu Mealy'ego w równoważny mu układ Moore'a, zastosujemy następujące przyporządkowania: $A/0=B_1$, $C/1=B_2$, $C/0=B_3$, $D/0=B_4$. W rezultacie otrzymamy tablice i graf stanów-wyjść przedstawiony na rys. 16.22d (na rys. 16.22a, b i c podaliśmy grafy i tablice Moore'a odpowiadające grafom i tablicom odpowiednio z rys. 16.20a, b, c).

Zauważmy, że w przypadku układu Mealy'ego kolejne kroki syntezy realizujemy dla tablicy z rys. 16.20d (jako minimalnej). Dla układu Moore'a minimalne są dwie tablice z rys. 16.22c, d (różniące się tylko oznaczeniami).

16.5.4 Kodowanie tablicy stanów-wyjść

Kodowaniem stanów nazywamy wzajemnie jednoznaczne przyporządkowanie symbolicznym stanom układu sekwencyjnego liczb binarnych (kodów stanów).

Przypomnijmy, że sposób kodowania stanów wewnętrznych ma wpływ na złożoność układów kombinacyjnych, realizujących zarówno funkcje wzbudzeń jak i funkcje wyjść, liczbę elementów pamięciowych (przerzutników), szybkość działania układu (układ szeregowy, równoległy) oraz niezawodność jako konsekwencja liczby użytych elementów i sposobu ich połączenia. Czy należy zatem syntezyować układ sekwencyjny dla wszystkich teoretycznie możliwych kodowań n stanów za pomocą liczb k -bitowych? Zauważmy, że tych kodowań jest

$$\frac{2^k!}{(2^k - n)!}$$

(liczba wariacji bez powtórzeń z 2^k elementów po n elementów), czyli przykładowo dla $n=5$ i $k=3$ liczba kodowań wynosi 6720 ale już dla $n=10$ i $k=4$ liczba kodowań wzrasta do $2.91 \cdot 10^{10}$.

Okazuje się jednak, że wiele kodowań daje identyczną złożoność układu kombinacyjnego wzbudzającego wejścia informacyjne przerzutników D. Nietrudno sprawdzić samemu na przykładzie czterech stanów, które zakodujemy dowolnie za pomocą czterech zestawów wartości dwóch zmiennych stanu, następnie zmienimy kodowanie poprzez dopełnienie wartości pierwszej zmiennej stanu, czy wreszcie zamienimy miejscami kolumny kodujące stany, to we wszystkich wymienionych wyżej przypadkach funkcje wzbudzeń przerzutników mają identyczną złożoność. Dlatego liczba istotnie różnych wariantów kodowań, czyli takich, które prowadzą do istotnie różnych realizacji układowych, będzie $k!$ razy mniejsza od teoretycznie możliwych, czyli wyniesie [3]

$$\frac{(2^k - 1)!}{(2^k - n)!k!}$$

gdzie n - liczba stanów, a k - liczba zmiennych stanu. Zatem dla powyższych przykładów dla których $n=5$, $k=3$ liczba istotnie różnych kodowań wyniesie 140 (w porównaniu z 6720) a dla $n=10$ i $k=4$ już tylko 75675600.

Wydaje się jednak, że przypadków do zsyntezyowania jest i tak za dużo i nie obejdzie się bez systemów wspomagania komputerowego. Najlepiej byłoby skorzystać z metod, które bez uciekania się do syntezy układu umożliwiłyby wybranie optymalnego wariantu kodowania. Trudność opracowania takich metod wynika przede wszystkim ze złożoności kryterium doboru wariantu kodowania, które powinno uwzględniać np. rodzaj używanych elementów (bramek, przerzutników), strukturę połączeń, rodzaj układu itp. Z drugiej strony zdarza się i tak, że wraz ze zwiększeniem liczby przerzutników ponad niezbędną, upraszcza się część kombinacyjna układu (porównaj układy z rys. 16.10 i 16.13).

Większa liczba przerzutników może zaś nie mieć znaczenia, ponieważ zazwyczaj w jednym układzie scalonym umieszcza się po kilka przerzutników. Między innymi z tego powodu proces kodowania wykorzystywany w podręczniku sprowadzimy do reguł heurystycznych.

Reguła I

1. Dwa stany poprzednie w stosunku do rozważanego stanu należy kodować za pomocą liczb binarnych różniących się tylko na jednej pozycji.
2. Dwa stany, które są stanami następnymi danego stanu, należy kodować za pomocą liczb binarnych różniących się na jednej pozycji.

Przytoczona reguła obowiązuje tylko dla układów sekwencyjnych wykorzystujących przerzutnik D. Nietrudno zauważyć, że spełnienie obu reguł może w konkretnych przypadkach prowadzić do sytuacji konfliktowych (reguła nie może być spełniona). Należy wtedy przyjąć rozsądny kompromis (np. najmniejsza liczba konfliktów).

a)

$y_1^t y_2^t \backslash x^t$	0	1
A=00	00/0	01/0
C=01	10/0	01/0
—	—	—
D=10	00/0	01/1

$y_1^{t+1} y_2^{t+1} / Z^t$

układ Mealy'ego

b)

$y_1^t y_2^t \backslash x^t$	0	1	Z^t
$B_1=00$	00	10	0
$B_2=01$	11	10	1
$B_4=11$	00	01	0
$B_3=10$	11	10	0

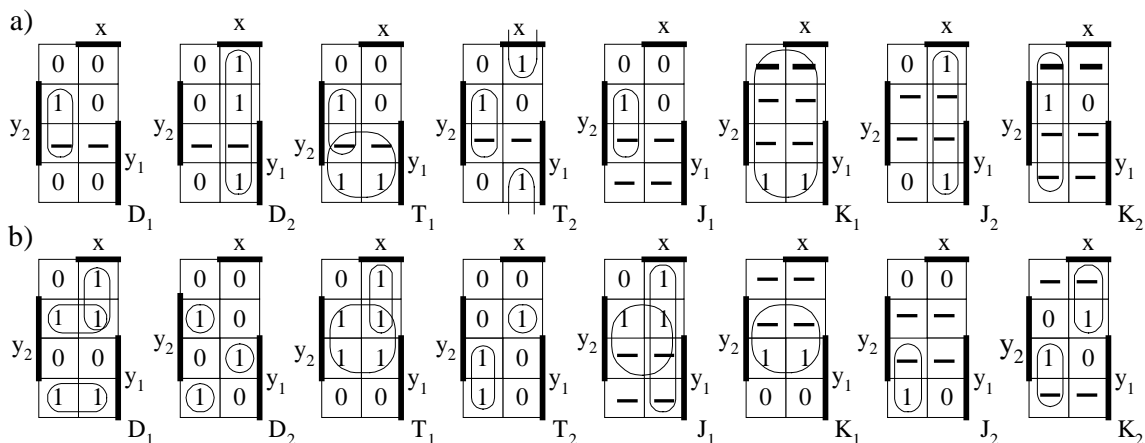
$y_1^{t+1} y_2^{t+1}$

układ Moore'a

Rys.16.23 Tablice przejść-wyjść układu rozpoznającego sekwencję 101

Reguła II

1. Stan z dużą liczbą zbiegających się do niego przejść (strzałek w grafie stanów) kodujemy zestawem zawierającym możliwie dużo zer.
2. Dwa stany, między którymi jest dużo przejść, powinny być kodowane binarnymi zestawami różniącymi się tylko na jednej pozycji (sąsiednimi).

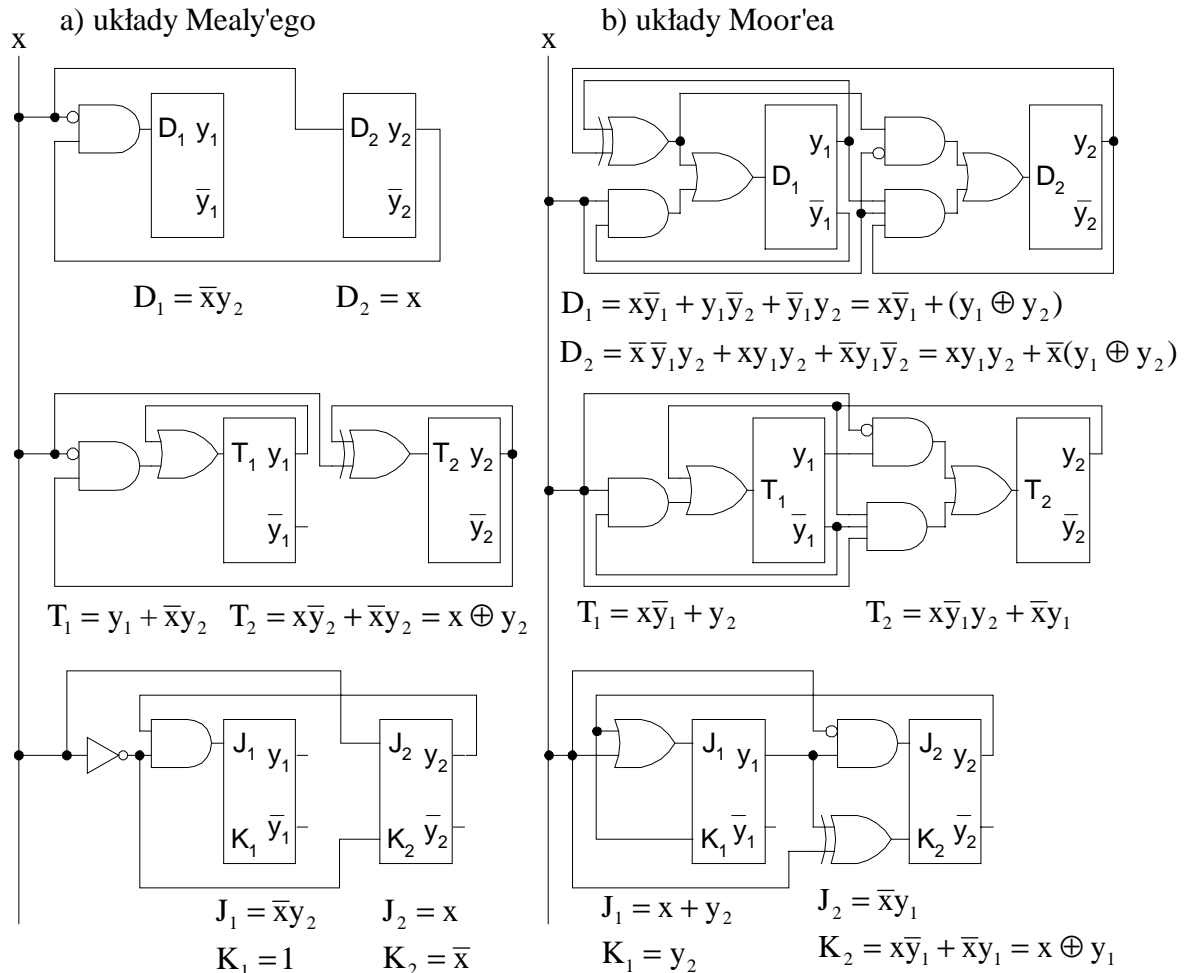


Rys.16.24 Funkcje wzbudzeń dla tablic przejść z rys. 16.23: a - układ Mealy'ego, b - Moore'a

Przyjmując, że stan początkowy kodować będziemy za pomocą samych zer, zgodnie z regułą II tablice stanów-wyjść możemy zakodować tak, jak to podaliśmy na rys. 16.23.

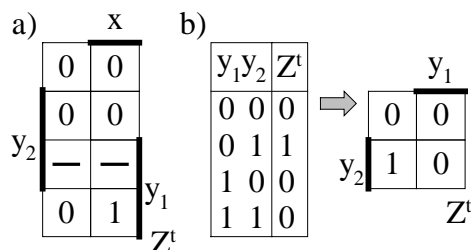
16.5.5 Układ realizujący funkcje wyjść

Wróćmy do tablic przejść-wyjść układu rozpoznającego sekwencje 101, które przedstawiliśmy na rys. 16.23. Tablica ta wyznacza dwie funkcje - przejść i wyjść. Realizacje układowe funkcji przejść, wykorzystujące różne rodzaje przerzutników podaliśmy na rys. 16.25 (funkcje wzbudzeń przedstawiliśmy na rys. 16.24). Zaznaczmy w tym miejscu, że wybór różnych przerzutników ma jedynie pokazać ich wpływ na złożoność układu, jednak w praktyce wybieramy przerzutniki jednego rodzaju, zwykle D lub JK.



Rys. 16.25 Realizacja tablic przejść z rys. 16.24 (wejścia zegarowe pominięto)

Pozostaje nam jedynie dołączenie układów realizujących funkcje wyjść. Na rys. 16.26a,b przedstawiliśmy tablice Karnaugh z powyższymi funkcjami wyjść, odpowiednio dla układu Mealy'ego i Moore'a.



Rys. 16.26 Funkcje wyjść: a - układ Mealy'ego, b - układ Moore'a

Funkcje wyjść są następujące:

układ Mealy'ego - $Z = xy_1$,

układ Moore'a - $Z = \bar{y}_1y_2$.

Układy wyjść możemy dorysować do dowolnego z układów przejść z rys. 16.25a (układ Mealy'ego) i rys. 16.25b (układ Moore'a).

Poprzez odpowiednie zakodowanie stanów staraliśmy się zmniejszyć złożoność układu przejść, to

jednak w ogólnym przypadku, kodowanie powinno uwzględnić złożoność układu wyjść, spodziewając się mniejszej łącznej złożoności obu układów. Przykładowo, gdy syntezujemy jednowyjściowy układ Moore'a, moglibyśmy, jeżeli jest to tylko możliwe, tak dobrać kodowanie by jedna z kolumn wartości pewnej zmiennej stanu była identyczna z kolumną funkcji wyjścia.

16.5.6 Stan początkowy układu

Układ synchroniczny rozpoznający sekwencję 101 będzie działał poprawnie, jeżeli po włączeniu zasilania znajdzie się w stanie początkowym A (rys. 16.23a). Zdarza się, że układ synchroniczny po włączeniu zasilania, losowo przechodzi do dowolnego stanu. W szczególności może to być stan, z którego nie ma przejścia do stanu następnego. Jeśli natomiast układ z rys. 16.23a po włączeniu zasilania przeszedłby do stanu C, to po podaniu sekwencji 01 sygnalizowałby rozpoznanie sekwencji 101, ponieważ $z=1$. Dlatego sytuacje powyższe są niepożądane.

Należy zatem w taki sposób zaprojektować układ, aby po włączeniu zasilania następowało automatyczne „zerowanie” wszystkich przerzutników (zakładając, że stan początkowy kodowany jest 00...0). Do tego celu możemy wykorzystywać ustawiające wejścia asynchroniczne przerzutników (inicjowanie asynchroniczne) bądź bramkując sygnał zegarowy z sygnałem inicjującym (inicjowanie synchroniczne).

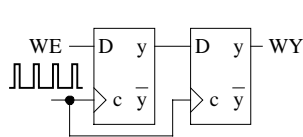
16.5.7 Zasady synchronizacji

Układ synchroniczny będzie działał poprawnie, jeżeli tylko sygnały wejściowe nie zmieniają swojej wartości podczas zmiany stanu, czyli muszą być one stabilne w przedziale czasu wyznaczonym przez czas wyprzedzenia (*set-up*) i utrzymania (*hold*).

Powyższe sformułowanie nie wymaga większego uzasadnienia, jeżeli tylko uważnie zapoznaliśmy się z dotychczasową treścią niniejszego rozdziału. Istnieją jednak dodatkowe zasady dotyczące synchronizacji, których spełnienie ma bezpośredni wpływ na poprawne działanie układu synchronicznego. Postaramy się krótko je przybliżyć.

Rozważmy kaskadowo połączone dwa przerzutniki D (rys. 16.27). Aby układ taki działał poprawnie, muszą być spełnione przynajmniej dwa warunki:

1. przed zmianą stanu na wejściach obu przerzutników są podane poprawne wartości,
2. każdy z przerzutników nie powinien zmienić swego stanu więcej niż jeden raz w czasie trwania wyróżnionej wartości sygnału zegarowego.



Rys. 16.27 Kaskadowo połączone przerzutniki D

Dodatkowo oba przerzutniki powinny reagować na to samo zbocze zegara. Gdyby pierwszy przerzutnik reagował na zbocze wznoszące a drugi na opadające, to układ ten działałby niepoprawnie. W ogólnym przypadku nie jest dobrą praktyką projektową wykorzystywanie w tym samym układzie przerzutników reagujących na różne zbocza.

Budując układ z rys. 16.27 zakładamy, że wyróżniona wartość sygnału zegarowego propaguje się do obu przerzutników w tym samym czasie oraz czasy propagacji przez przerzutniki są jednakowe (lub porównywalne). Otóż takie założenie nie jest słuszne. Jeżeli przyjmiemy, że pierwszy przerzutnik reaguje na tyle szybko, że zmieni swój stan a drugi z nich ("wolny") nie zdąży zareagować na poprzednią wartość wyjścia pierwszego przerzutnika to układ przestanie działać poprawnie. Podobnie się stanie, gdy wyróżniona wartość zegara dotrze do pierwszego przerzutnika szybciej niż do drugiego (co może być spowodowane bardzo dużym opóźnieniem na przewodzie doprowadzającym sygnał zegara do drugiego przerzutnika).

Zjawisko to nazywane asynchronizmem zegara (*clock skew*), jest poważnym problemem w układach pracujących z dużymi częstotliwościami sygnału zegarowego, gdy jego długość fali jest porównywalna z odległościami między elementami układu. Ponieważ współczesne układy logiczne pracują z coraz większymi częstotliwościami, to oprócz problemów z odprowadzeniem ciepła, należy dodatkowo układowo eliminować asynchronizm zegara. Problem jest na tyle poważny, że coraz częściej są podejmowane próby zastąpienia np. mikroprocesorów synchronicznych, mikroprocesorami asynchronicznymi, w których nie ma sygnału zegarowego [4].

Jeżeli tylko "przesunięcie" między zboczami sygnałów zegarowych będzie mniejsze od różnicy czasu propagacji przez przerzutnik i czasu utrzymania lub szybciej dotrze sygnał zegara do drugiego a potem do pierwszego przerzutnika (odwrotnie niż wyżej), to układ będzie działał poprawnie.

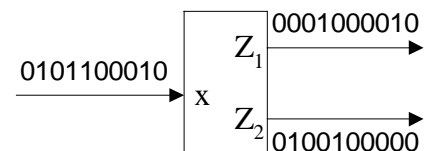
Spróbujmy oszacować najmniejszy odcinek czasu pomiędzy kolejnymi zboczami zegara jaki musi być zapewniony aby układ działał poprawnie. Intuicyjnie wyczuwamy, że nie można w "nieskończoność" zwiększać częstotliwości zegara, ponieważ przy pewnej granicznej częstotliwości układ przestanie działać poprawnie. Dla rozważanego układu z rys. 16.27 minimalny odcinek czasu pomiędzy kolejnymi zboczami zegara, powinien być większy niż suma czasu opóźnienia lewego przerzutnika i czasu wyprzedzenia prawego przerzutnika (plus suma opóźnień na bramkach, które mogłyby się znajdować pomiędzy wyjściem pierwszego a wejściem drugiego przerzutnika).

Literatura

- [1] Mealy G.H., A method for synthesizing sequential circuits, Bell Sys. Tech. J., vol.34, September 1955, str. 1045-1079,
- [2] Moore E.F., Gedanken - experiments on sequential machines, Automata Studies, Annals of Mathematical Studies, no.34, Princeton University Press, 1956, str.129-153,
- [3] Rhyne V.T., Noe P.S., On the number of distinct state assignments for a sequential machine, IEEE Trans. Comp., C-26, str. 73-75, 1977
- [4] Kaliś A., Układy asynchroniczne – ciekawa alternatywa, VIII Konferencja Systemy Czasu Rzeczywistego, Krynica, 24-27 września 2001, str. 297-310.

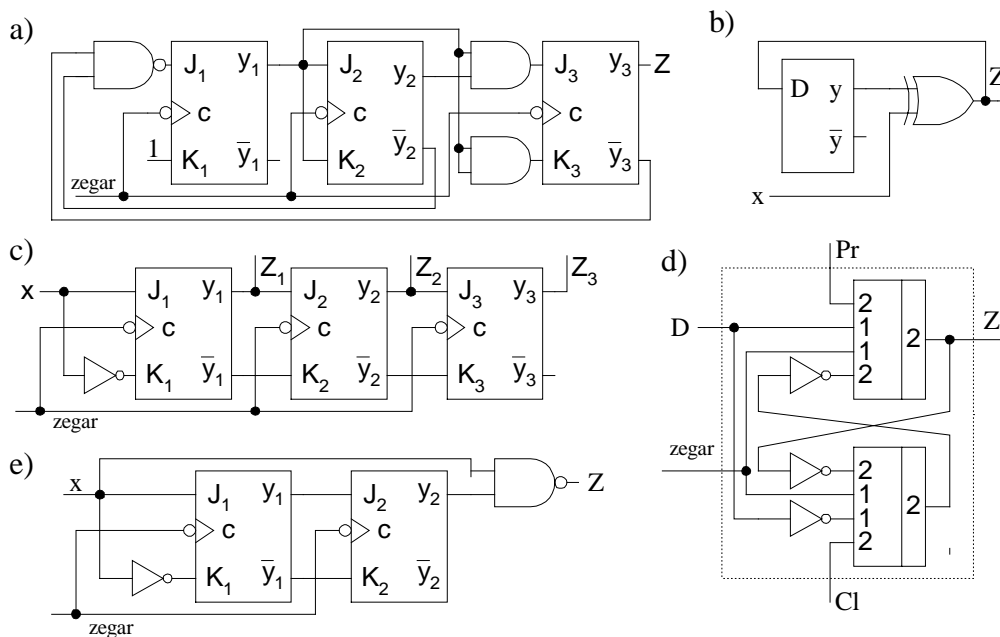
ĆWICZENIA

1. Zaprojektuj układ synchroniczny o jednym wejściu x i dwóch wyjściach z_1 oraz z_2 działający w następujący sposób: pierwsza jedyńska napotkana w sekwencji wejściowej generuje $z_1=1$, druga kolejna jedyńska $z_2=1$, trzecia jedyńska - $z_1=1$, czwarta jedyńska - $z_2=1$ itd. Dodatkowo należy przyjąć, że $z_1 z_2 = 0$ oraz gdy $x=0$, to $z_1=0$ i $z_2=0$ (patrz ilustracja na rys. 16.28, gdzie w chwili $t_1; x=0$, $t_2; x=1$, $t_3, t_4, t_5; x=0$, $t_6; x=1$, itd).
2. Przeanalizuj układy synchroniczne podane na rys. 16.29. Podaj grafy przejść i tablice przejść wyjść. Wskaż układy Mealy'ego, Moore'a oraz autonomiczne. Literą Z oznaczono wyjście.
3. Zaprojektuj układ synchroniczny sygnalizujący na wyjściu, że w sekwencji wejściowej wystąpiły przynajmniej trzy serie dwóch jedynek.
4. Zaprojektuj układ synchroniczny, który generuje na wyjściu Z jedynekę, jeżeli podamy na wejście układu dowolną sekwencję o długości podzielnej przez 3, składającą się z liter a i b. Długością sekwencji nazywamy liczbę liter wejściowych w sekwencji.



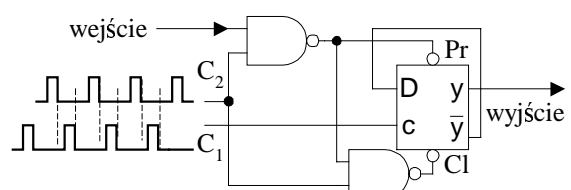
Rys.16.28 Ilustracja do zadania

5. Zaprojektuj układ synchroniczny zliczający impulsy zegarowe (modulo 4), gdy sygnał sterujący $s=1$, oraz pozostający w aktualnym stanie gdy $s=0$.
6. Zaprojektuj układ synchroniczny porównujący ze sobą dwie sekwencje wejściowe. Podaj dwa warianty rozwiązania: jedno dla sekwencji binarnych wprowadzanych od najstarszego bitu, drugie dla sekwencji wprowadzanych od bitu najmłodszego.
7. Czy podany niżej algorytm znajduje maksymalny zbiór stanów zgodnych? Jeżeli tak - podaj jego zalety i ewentualne wady w stosunku do podanych wyżej; jeżeli nie - wskaż błąd;
 - a) Kolumny tablicy trójkątnej ponumerowane są poczynając od lewej strony kolejnymi liczbami $1, 2, \dots, j, \dots, m, \dots$. $M = \{C_1, C_2, \dots\}$ oznacza zbiór, którego elementy stanowią zbiory stanów zgodnych. Wybierz z prawej strony tablicy pierwszą kolumnę, w której nie wszystkie kratki są wykreślone. Załóżmy, że kolumna ta odpowiada stanowi q_m . Do zbioru M wprowadź wszystkie pary stanów zgodnych postaci $\{q_m, q_j\}$, $j > m$. (Czy teraz zbiór M zawiera zbiory zgodne postaci $\{q_m, q_j\}$, $j > m$?)
 - b) Przejdź do kolejnej niecałkowicie wykreślonej kolumny o niższym numerze, np. k . Utwórz zbiór S_k zawierający pary zgodne postaci $\{q_j, q_k\}$ $j > k$.
 - c) Dla $\forall C_i \in M$ utwórz zbiór $M' = \{C_i'\}$, $C_i' = (S_k \cap C_i) \cup \{q_k\}$. Jeżeli dla pewnego $q_i \in S_k$ nie istnieje C_m' taki, że $\{q_i, q_k\} \subseteq C_m'$, to włącz $\{q_i, q_k\}$ do M' . (Czy teraz zbiór M' zawiera wszystkie zbiory stanów zgodnych zawierające q_k ?)
 - d) Oznacz $M \cup M'$ jako M . Jeżeli dla $\forall C_i, C_j \in M$, $C_i \subseteq C_j$, to wykreśl C_i z M (Czy teraz zbiór M zawiera wszystkie zbiory zgodne postaci q_k, q_{k+1}, \dots, q_n ?).
 - e) Powtórz kroki b, c, d przemieszczając się w lewo aż do wyczerpania kolumn tablicy.
 - f) Zbiór M jest maksymalnym zbiorem stanów zgodnych.



Rys. 16.29 Przykładowe układy synchroniczne

8. Podaj zasadę działania układu z rys.16.30. Rozpocznij od podania na wejście układu następującej sekwencji:000111000111000....
9. Czy układ z rysunku 16.18 (kodowanie I) będzie działał poprawnie, jeżeli wiadomo, że przerzutnik D zmienia swój stan w okresie narastania zbocza sygnału zegara, zaś przerzutnik T w okresie opadającego zbocza?
10. Na grafie z rysunku 16.12d widzimy, że do stanu D nie dochodzą strzałki. Jakie znaczenie posiada stan D dla układu?
11. Jaki jest czas utrzymania dla dwutaktowego przerzutnika JK?



Rys. 16.30 Przykładowy układ

12. Jaka jest maksymalna częstotliwość, przy której układ z rys. 16.27 będzie jeszcze działał poprawnie. Przyjmijmy, że wykorzystujemy przerzutniki: a) 74S74, b) 74LS74.

17 ELEMENTY DIAGNOSTYKI UKŁADÓW SYNCHRONICZNYCH

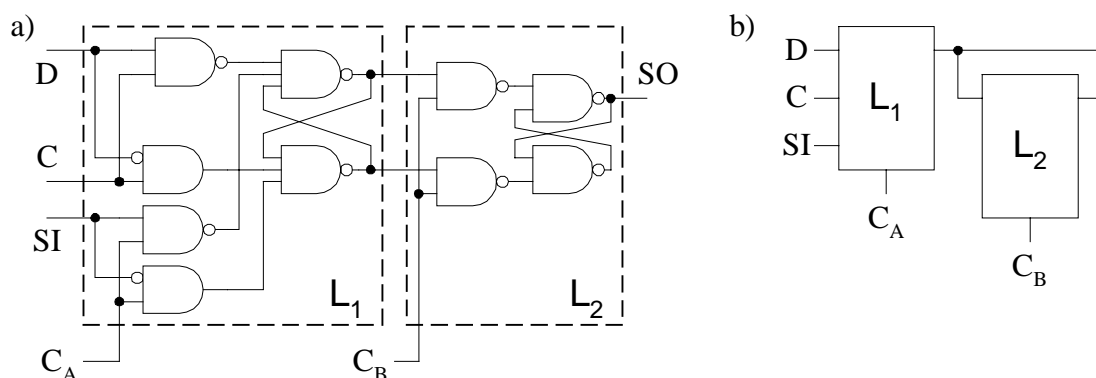
Wykrywanie uszkodzeń w układach synchronicznych jest o wiele bardziej złożone niż w układach kombinacyjnych. Obserwowalne wartości wyjść układu sekwencyjnego zależą nie tylko od jego wartości wejściowych, lecz również od stanu wewnętrznego, reprezentowanego przez sygnały na liniach sprzężeń zwrotnych. Problem wyszukiwania testów, jak i sam proces testowania komplikuje fakt, że w odróżnieniu od układów kombinacyjnych, gdzie testem jest odosobniony zestaw wartości zmiennych wejściowych, dla układu sekwencyjnego testem zazwyczaj musi być sekwencja pobudzeń. Dlatego powszechnie uznaje się pogląd, że problemy wyszukiwania testów dla układów kombinacyjnych są stosunkowo mniej skomplikowane niż dla układów sekwencyjnych.

Synchroniczne układy sekwencyjne (tylko nimi będziemy się zajmować) to nic innego jak układ kombinacyjny objęty pętlami sprzężeń zwrotnych. Naturalne wydaje się podejście, by metody testowania układów kombinacyjnych przenieść na układy sekwencyjne. Układ sekwencyjny powinien być zatem specjalnie zaprojektowany tak, by mógł być łatwo testowalny. Ogólnie, układ o wysokim stopniu testowalności, to taki układ, który ma następujące cechy:

1. daje się w prosty sposób sprowadzić do dowolnego stanu początkowego (inicjalizacja),
2. możemy osiągnąć dowolny stan wewnętrzny układu przykładając do jego wejść prostą sekwencję zestawów wartości zmiennych wejściowych,
3. stan wewnętrzny układu można jednoznacznie i prosto identyfikować na jego wyjściach (lub specjalnych punktach testowania).

Pierwsze dwie cechy są pewną miarą sterowalności danego układu. Trzecia cecha jest miarą obserwowalności. Dla przykładu, układy kombinacyjne mają idealną sterowalność i obserwowalność, ponieważ nie mają one stanów wewnętrznych.

Układy sekwencyjne o wysokim stopniu testowalności należy zazwyczaj odpowiednio skonstruować. Różne stosuje się podejście i rozwiązania. My zajmiemy się tylko jednym, wykorzystywanym w produktach firmy IBM. Istota podejścia sprowadza się do odpowiedniego zaprojektowania pamięci układu sekwencyjnego oraz wykorzystania testów otrzymanych dla części kombinacyjnej (układu przejść).

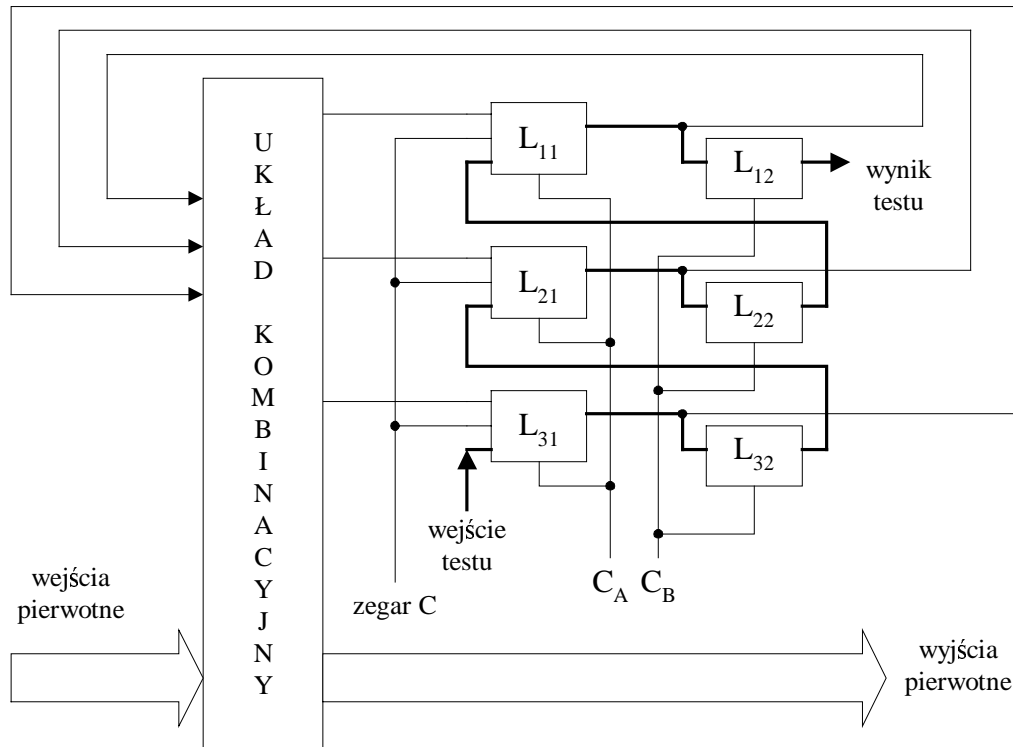


Rys. 17.1 Nietypowe elementy pamiętające: a - schemat szczegółowy, b - blokowy

Podstawowym elementem pamięci są pary przerzutników wyzwalanych poziomem L_1 i L_2 . Przerzutniki te nazywamy zatrzaskami (*latch*). Schemat logiczny i sposób powiązania ze sobą przerzutników L_1 i L_2 przedstawiliśmy na rys. 17.1. Pierwszy przerzutnik ma dwie pary wejść. Para D i C stanowi odpowiednio wejście informacyjne i normalne wejście zegarowe, zaś para SI i C_A służy do przesuwania informacji pamiętanej w przerzutnikach. Tylko jeden z

sygnałów zegarowych C i C_A może być aktywny w danej chwili.

Przykładowy schemat blokowy układu sekwencyjnego z trzema zmiennymi stanu oraz zasadę łączenia par przerzutników L_1 i L_2 pokazaliśmy na rys. 17.2. Przesunięcia danych w łańcuchu przerzutników (pogrubiona kreska na rys. 17.2) następuje poprzez włączenie i wyłączenie sygnału zegara C_A (wartość I zapamiętuje się w L_1). Następnie wartość pamiętana w L_1 przesyłana jest do L_2 poprzez włączenie i wyłączenie zegara C_B .



Rys. 17.2 Układ sekwencyjny o wysokim stopniu testowalności

W czasie normalnej pracy układu, tzn. w czasie, gdy realizuje on funkcję zgodną z tablicą przejść-wyjść, wykorzystywany jest tylko przerzutnik L_1 (jest to najprostszy tryb pracy).

Procedura testowania części kombinacyjnej tak skonstruowanego układu jest następująca:

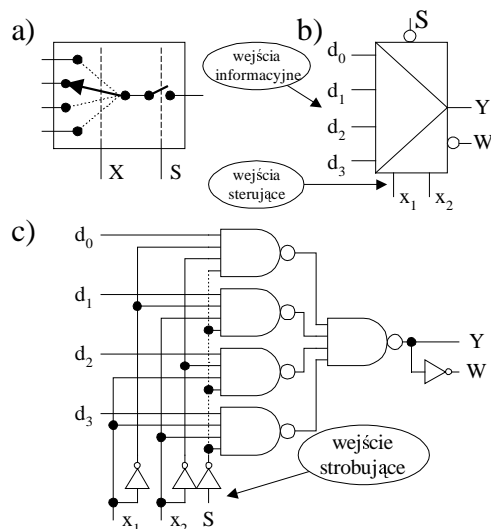
1. Sprawdzamy poprawność działania rejestru przesuwającego ($C=0$ oraz sterujemy układem za pomocą nie zachodzących na siebie impulsów zegarów C_A i C_B).
2. Wsuwamy do rejestru wartości stanowiące tę część testu, którą podajemy na wejście Y , a pozostałą część na wyjścia pierwotne X .
3. Włączamy na jeden cykl zegar C , test $T=(X;Y)$ oddziałuje na układ kombinacyjny. Wynikiem oddziaływania testu są wartości zapamiętane w rejestrze przesuwającym oraz na wyjściach pierwotnych, które możemy obserwować.
4. Wysuwamy wynik testu z rejestru przesuwającego.
5. Powtarzamy kroki 2-4 aż do wyczerpania wszystkich testów wykrywających uszkodzenia części kombinacyjnej.

Nietrudno zauważyć, że omówiony sposób postępowania powoduje, że testowanie synchronicznego układu sekwencyjnego przebiega podobnie jak testowanie układu kombinacyjnego, zatem upraszcza się proces wyznaczania testów.

18 WYBRANE UKŁADY TYPOWE

18.1 Multiplexer i demultiplexer

Multiplexer (*multiplexer*, *MUX*) jest układem kombinacyjnym o jednym wyjściu Y , r wejściach sterujących x oraz 2^r wejściach informacyjnych d . Często występuje jeszcze dodatkowe wejście bramkujące S (nazywane też strobojącym lub zezwalającym). Jego zadaniem jest połączenie jednego wejścia informacyjnego (wskazanego przez zmienne sterujące) z wyjściem, czyli multiplexer przenosi sygnał ze wskazanego wejścia na wyjście.



Rys. 18.1 Multiplexer, a - poglądowa ilustracja działania multiplexera, b - oznaczenie, c - układ (na bramkach NAND)

Zasadę pracy multiplexera poglądowo przedstawiliśmy na rys.18.1a. Oznaczenie oraz przykład czterowejsiowego multiplexera pokazaliśmy na rys.18.1b oraz c. Sygnał bramkujący S (nazywany też strobojącym bądź zezwalającym) zaznaczyliśmy linią przerywaną (nie umniejsza to jednak istotnemu znaczeniu tego sygnału!).

Funkcja opisująca multiplexer z rys. 18.1c jest następująca ($S=1$):

$$Y = \bar{x}_1 \bar{x}_2 d_0 + \bar{x}_1 x_2 d_1 + x_1 \bar{x}_2 d_2 + x_1 x_2 d_3$$

Jeżeli na wejścia sterujące $X=(x_1, x_2)$ podamy pewien zestaw wartości, to „wybrane” zostanie wejście d_i , gdzie i jest odpowiednikiem dziesiętnym zestawu wartości wejścia sterującego oraz wartość d_i zostanie przeniesiona na wyjście Y .

Rozważmy układ multiplexera, którego wejścia d_i , $0 \leq i \leq 3$, nie będą traktowane jako niezależne, ale na każde z nich będziemy mogli podać sygnały pochodzące jedynie ze zbioru $\{0, 1, x_3, \bar{x}_3\}$. Wtedy funkcja wyjścia multiplexera jest następująca:

$$Y(x_1 x_2 x_3) = \bar{x}_1 \bar{x}_2 Y_0(x_3) + \bar{x}_1 x_2 Y_1(x_3) + x_1 \bar{x}_2 Y_2(x_3) + x_1 x_2 Y_3(x_3)$$

gdzie: $Y_0(x_3), \dots, Y_3(x_3) \in \{0, 1, x_3, \bar{x}_3\}$

Powyższa postać jest rozwinięciem funkcji Y ze względu na zmienne x_1 oraz x_2 . Jeżeli dobierzemy wartości funkcji resztowych ze zbioru $\{0, 1, x_3, \bar{x}_3\}$, to możemy wygenerować wszystkie funkcje przełączające trzech zmiennych (czyli 256).

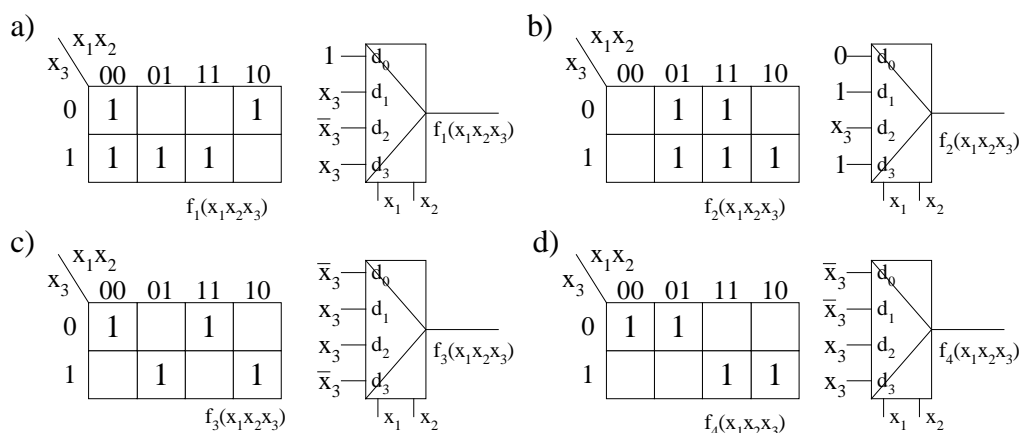
Przykład:

Niech będzie dana funkcja $f(x_1 x_2 x_3) = \Sigma(0, 1, 3, 4, 7)$. Wpisując funkcję do tablicy Karnaugh'a zauważymy, że zmienne sterujące x_1 i x_2 wskazują kolumny tablicy. Wystarczy zatem na wejście d_0 podać stałą wartość 1 (w kolumnie $x_1 x_2 = 00$ są same jedynki), na wejście d_1 zmienną x_3 (wartości funkcji w kolumnie 01 są równe wartościom zmiennej x_3), na wejście d_2 dopełnienie zmiennej \bar{x}_3 (wartości funkcji w kolumnie 10 są równe \bar{x}_3) oraz na wejście d_3 zmienną x_3 . Łatwo też sprawdzić, że:

$$\begin{aligned} f(x_1 x_2 x_3) &= \bar{x}_1 \bar{x}_2 (1) + \bar{x}_1 x_2 (x_3) + x_1 \bar{x}_2 (\bar{x}_3) + x_1 x_2 (x_3) = \\ &= \bar{x}_1 \bar{x}_2 \bar{x}_3 + \bar{x}_1 \bar{x}_2 x_3 + \bar{x}_1 x_2 \bar{x}_3 + x_1 \bar{x}_2 \bar{x}_3 + x_1 x_2 x_3 = \Sigma(0, 1, 3, 4, 7) \end{aligned}$$

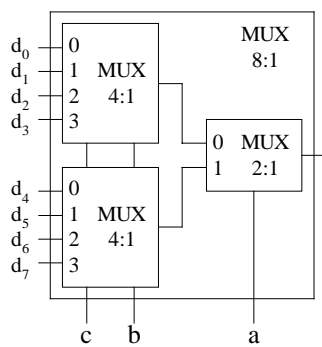
Realizację powyższej funkcji z wykorzystaniem multiplexera przedstawiliśmy na rys. 18.2a. Kolejne przykłady realizacji wybranych funkcji pokazaliśmy na rys.18.2b,c oraz d.

Funkcje przełączające czterech i pięciu zmiennych możemy realizować wykorzystując, odpowiednio, 8- i 16-wejściowe multiplexery bądź łącząc wiele multiplexerów 4-wejściowych[1]. Przykładową realizację 8-wejściowego multiplexera pokazaliśmy na rys. 18.3.



Rys. 18.2 Przykłady realizacji funkcji przełączających z wykorzystaniem multiplexera

Podsumowując, należy stwierdzić, że jedną z podstawowych zalet multiplexera jest możliwość realizacji dowolnej funkcji n zmiennych bez uciekania się do jej minimalizacji i realizacji za pomocą wielu podstawowych bramek logicznych, co ma istotne znaczenie dla funkcji większej liczby zmiennych. Jednak multiplexery są wykorzystywane przede wszystkim do sterowania przepływem danych z wybranego źródła do wyjścia. Ponieważ źródłem danych nie musi być pojedyncza linia sygnałowa ale ich grupa, dlatego na rys. 18.4 przedstawiliśmy ogólną strukturę multiplexera.



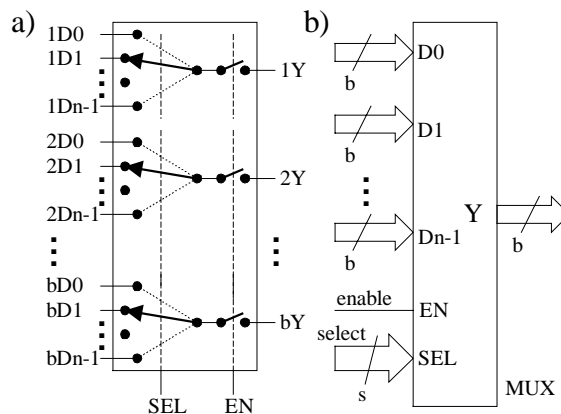
Rys. 18.3 Składanie

Mechaniczny odpowiednik działania multiplexera pokazaliśmy na rys. 18.4a, który jest równoważny multiplexerowi z rys. 18.4b za wyjątkiem tego, że ten ostatni jest jednokierunkowy, czyli dane przesyłane są jedynie od źródła do wyjścia (mechaniczny odpowiednik sugerować mógłby dwukierunkowe przenoszenie danych).

Wyrażenie, które opisuje multiplexer jest następujące:

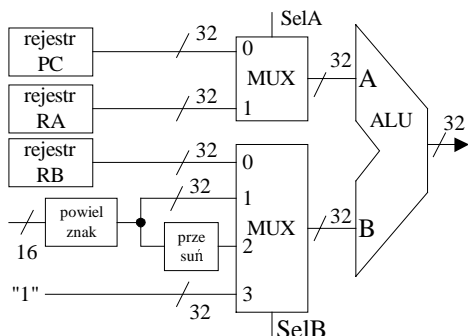
$$iY = \sum_{j=0}^{n-1} EN \cdot M_j \cdot iD_j$$

gdzie znak sumy oznacza sumowanie logiczne iloczynów. Zmienna iY (porównaj oznaczenia z rys. 18.4) reprezentuje pojedynczą wyjściową linię sygnałową ($1 \leq i \leq b$) a zmienna iD_j reprezentuje i -tą wejściową linię źródła j ($0 \leq j \leq n-1$), zaś M_j to iloczyn zupełny wejść sterujących SEL odpowiadający j . Zatem, gdy $EN=1$, wartość reprezentowana na wejściach SEL odpowiada j , to każda wyjściowa linia iY jest połączona z odpowiednią linią wybranej grupy wejściowej iD_j .



Rys. 18.4 Struktura multiplexera: a - poglądowy odpowiednik funkcjonalny, b - wejścia i wyjścia

Multiplexery są często wykorzystywane do precyzyjnego określenia przepływu danych w schematach układów logicznych. Przykładem może być fragment pewnego układu z rys. 18.5 pokazujący, które 32-bitowe dane zostaną podane na wejścia A i B jednostki arytmetyczno-logicznej (ALU). Odpowiednie dane są wybierane przez sygnały sterujące SelA oraz SelB.



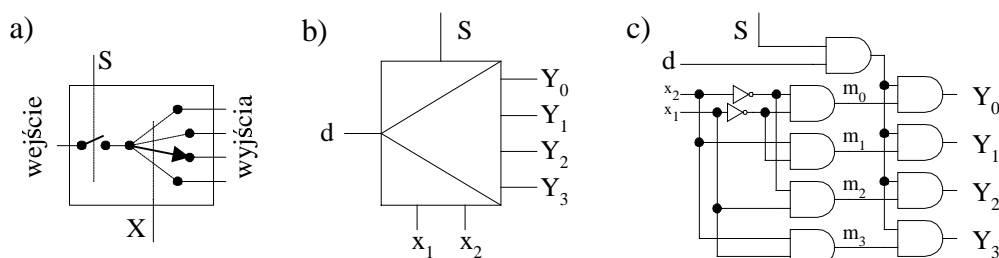
Demultiplexer (*demultiplexer*, *DEMUX*) jest układem kombinacyjnym o jednym wejściu informacyjnym d , r wejściach sterujących x oraz 2^r wyjściach Y .

Nazwa wskazywałaby na „odwrotne” działanie układu w stosunku do multiplexera. Oczywiście tak jest, ponieważ demultiplexer wybiera jedno z wyjść, na które skierowuje dane z wejścia (Rys. 18.6). Wyrażenie opisujące demultiplexer jest następujące:

$$Y_i = (m_i \cdot d) \cdot S$$

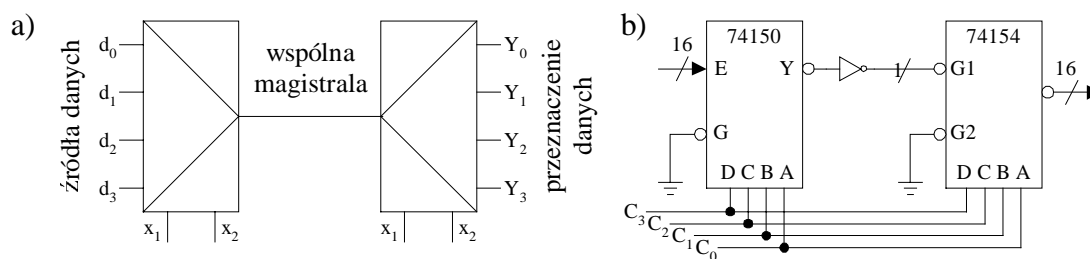
Rys. 18.5 Wykorzystanie multiplexerów

Wzajemne uzupełnienie się zachowania multiplexera i demultiplexera wykorzystuje się do sterowania przepływem danych między wybranym (przez multiplexer) źródłem danych a wybranym (przez demultiplexer) przeznaczeniem (Rys. 18.7a) a także do zmniejszenia liczby przewodów przy przesyłaniu na większe odległości (Rys. 18.7b).



Rys. 18.6 Demultiplexer: a - poglądowa ilustracja działania b - oznaczenie, c - realizacja układowa

Demultiplexer nazywany jest także dekoderym (*decoder*), ponieważ służy do przekształcania dowolnego kodu dwójkowego o długości r (wejścia sterujące) na kod „1 z 2^r ”, gdy $d=1$. Kodem „1 z n ” nazywamy zestaw dwójkowy o długości n , w którym tylko na jednej pozycji występuje jedynka a na pozostałych pozycjach wartości 0.



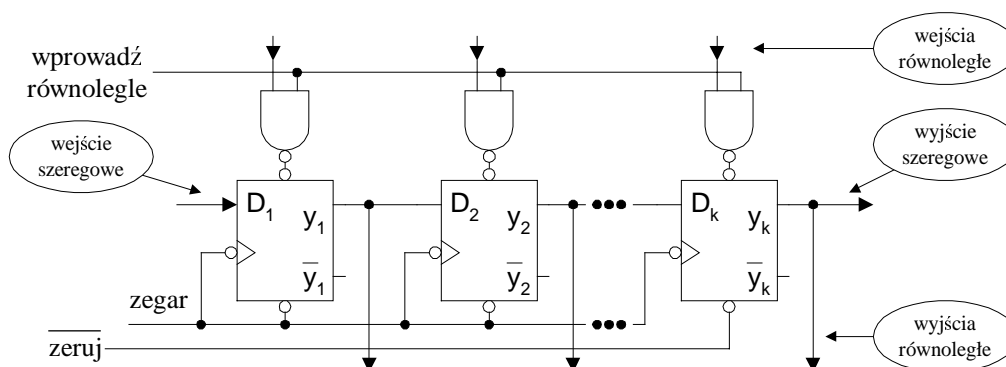
Rys. 18.7 Przykłady wykorzystania multiplexera i demultiplexera

Rozwiązanie z rys. 18.7a ma niewątpliwą zaletę polegającą na tym, że wybrane dane mogą być przesyłane poprzez magistralę danych (*data bus*), wspólną dla wszystkich urządzeń (źródeł i przeznaczeń danych). Złożoność układowa tego rozwiązania jest znacznie mniejsza od tego, które łączyłoby wszystkie urządzenia bezpośrednio ze sobą.

18.2 Rejestry

Do najważniejszych układów synchronicznych należą rejestry, które przede wszystkim służą do przechowywania danych binarnych. Dane mogą być wprowadzane do rejestru bit po bicie (szeregowo) bądź grupowo (równoległe).

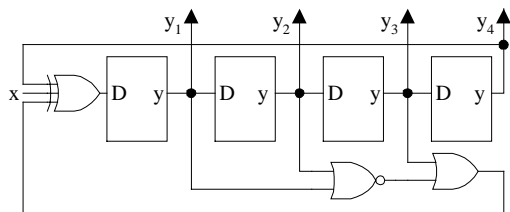
Jedną z podstawowych funkcji rejestrów jest operacja przesunięcia danych (rejestry przesuwające). Przesuwanie danych w rejestrze należy do podstawowych operacji wykonywanych w systemach cyfrowych. Wykorzystuje się przesuwanie w celu zamiany danych szeregowych na równoległe (i odwrotnie), mnożenia i dzielenia liczb binarnych przez liczbę równą potęgę dwójki, normalizacji liczb zmiennoprzecinkowych, itp. Rejestry mające możliwość przesuwania informacji w lewo bądź w prawo nazywamy rejestrami rewersyjnymi.



Rys. 18.8 Jednokierunkowy rejestr przesuwający

Przykładowy rejestr przedstawiliśmy na rys.18.8.

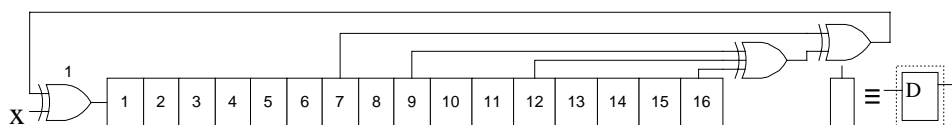
Podstawowa operacja wykonywana przez rejestr przesuwający sprowadza się do tego, że po każdym sygnale zegarowym ($i+1$) przerzutnik D zapamiętuje wartość pamiętaną w i -tym przerzutniku, przy czym dotyczy to wszystkich przerzutników oprócz pierwszego, który zapamiętuje wartość z wejścia szeregowego. Wartość pamiętana w prawym skrajnym przerzutniku jest gubiona (jeżeli nie jest pamiętana w jakimś innym urządzeniu).



Rys. 18.9 Generator wartości 4 zmiennych

W takim połączeniu - jeżeli wpisze tylko do jednego przerzutnika jedynkę, a do pozostałych zera - jedynka ta będzie krążyć w rejestrze tworząc ważny układ, tzw. wielofazowego generatora impulsów synchronizujących. Proponujemy prześledzenie zachowania rejestru przesuwającego, jeżeli z wejściem szeregowym połączymy wyjście dopełnione prawego skrajnego przerzutnika.

Jeżeli rejestr zaprojektujemy tak jak to pokazaliśmy na rys.18.9 ($n=4$), to otrzymamy układ generatora wszystkich zestawów zerojedynekowych o długości n .

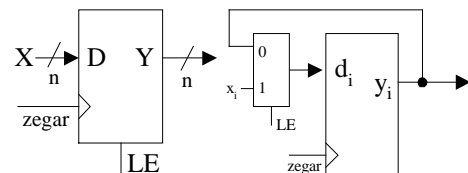


Rys. 18.10 Generator wykorzystywany w analizie sygnatur

Metoda testowania układów synchronicznych nazywana analizą sygnatur wykorzystuje rejestr przedstawiony na rys. 18.10. Wyjście testowanego układu doprowadza się do wejścia x i obserwuje sekwencję wyjściową na prawym skrajnym przerzutniku.

Proponuje się Czytelnikowi, by przeanalizował zachowanie układu, gdy $x = 0$ oraz sygnały doprowadzone do bramki nr 1 (EOR) pobierane są z innych przerzutników niż tych, które zaznaczyliśmy na rysunku (tzn. różnych od przerzutników 7, 9, 12 i 16).

Powyżej skupiliśmy uwagę na wykorzystaniu rejestru przesuwającego, jednak – o czym wspomnieliśmy na początku rozdziału – rejestr może służyć przede wszystkim jako urządzenie przechowujące informacje n -bitowe. Rejestr taki jest zestawem n przerzutników D, które zapamiętują dane z wejścia zewnętrznego tylko wtedy, gdy wystąpi aktywne zbocze zegara oraz ustawiony jest pewien sygnał zezwalający (*load enable*) $LE=1$. Gdy $LE=0$, informacja zapamiętana w przerzutnikach jest utrzymywana bez zmian. Stosowane oznaczenie oraz ilustracja zasady działania rejestru pamiętającego przedstawiliśmy na rys. 18.11.



Rys. 18.11 Rejestr pamiętający

18.3 Liczniki

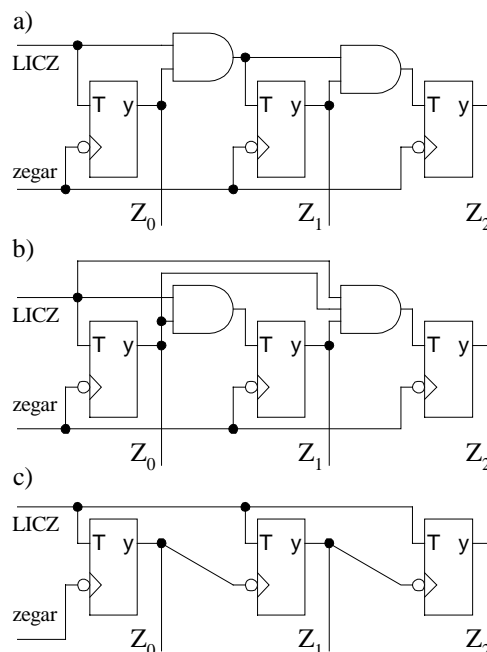
Licznikiem zwiemy układ sekwencyjny zliczający i zapamiętujący liczbę impulsów wejściowych w określonym przedziale czasu. Zliczane są zazwyczaj impulsy zegarowe.

Każdy licznik ma ograniczoną z góry „pojemność”, tzn. potrafi zliczyć określoną liczbę impulsów, po której przekroczeniu wraca do swojego stanu początkowego i zaczyna liczenie od początku. Licznik z powtarzającą się cyklicznie sekwencją stanów wewnętrznych nazywamy licznikiem modulo n (mod n), gdzie n oznacza liczbę różnych stanów wewnętrznych, po których licznik wraca do stanu początkowego i zliczając impulsy generuje identyczną sekwencję tych stanów.

Na rys. 18.12 przedstawiliśmy trzy liczniki modulo 8 (liczą impulsy wejściowe od 0 do 7).

Po każdym zboczu opadającym sygnału zegarowego licznik przechodzi do następnego stanu wewnętrznego (rys.18.12a). Wartość T_{i+1} jest iloczynem $T_i y_i$, gdzie T_i oraz y_i jest wejściem i wyjściem i -tego przerzutnika. Wartość T_0 otrzymujemy oddzielnie, ponieważ odnosi się do skrajnego lewego przerzutnika. Na wejście to podajemy sygnał LICZ. Jeżeli $LICZ=0$, to na wszystkie wejścia informacyjne przerzutników podajemy wartość zero, a tym samym żaden z przerzutników nie zmienia swojego stanu (patrz tablica wzbudzeń przerzutnika T). Gdy $LICZ=1$, wówczas na wszystkie wejścia informacyjne podajemy sygnał umożliwiający przejście do następnego stanu (Dlaczego?).

Przyjęcie układu sterującego pracą licznika za pomocą sygnału LICZ, który podaliśmy na rys. 18.12a powoduje spowolnienie jego pracy, ponieważ kolejne wystąpienie aktywnego zbocza zegara może wystąpić dopiero wtedy, gdy na



Rys. 18.12 Liczniki binarne modulo 8:

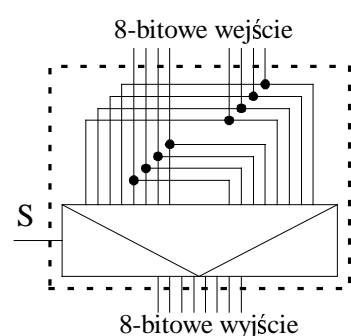
wszystkich wejściach T ustalą się poprawne wartości sygnałów. W szczególności dla n-pozycyjnego licznika następny impuls zegara może być podany po czasie w jakim ostatnia bramka z prawej strony ustali swoją wartość wyjściową, czyli po czasie $(n-1)*t$, gdzie t – opóźnienie jednej bramki AND. Szybkość zliczania zwiększymy, jeżeli układ bramek AND połączymy inaczej (rys.18.12b). Między chwilą ustalenia się następnego stanu przerzutników po podaniu impulsu zegara a gotowością nowych wartości na wejściach T powinien upłynąć jedynie czas opóźnienia jednej bramki AND (nie zaś $n-1$ bramek).

Ze względu na to, że stany przerzutników zmieniają się w takt impulsów zegarowych, a zmiana ta następuje „jednocześnie” na wszystkich przerzutnikach, dlatego liczniki te nazywamy równoległymi.

Na rys.18.12c pokazaliśmy licznik szeregowy. Jak wynika z rysunku, wszystkie wejścia zegarowe, oprócz skrajnego lewego przerzutnika, nie są połączone do wspólnej linii synchronizującej, ale z wyjściem sąsiedniego przerzutnika. Dlatego stan przerzutnika ulega zmianie w odpowiedzi na zmianę stanu wewnętrznego sąsiedniego przerzutnika.

18.4 Przesuwniki

Przesuwnikami (*shifters*) nazywamy układy **kombinacyjne**, których zadaniem jest przesunięcie wartości wejścia w prawo (w lewo) o zadaną liczbę pozycji. Operacja przesunięcia jest istotna w układach arytmetycznych, gdzie wymagane jest np. skalowanie (mnożenie lub dzielenie przez całkowitą potęgę dwójki). Ponieważ są to układy kombinacyjne, więc operacja przesunięcia wykonuje się znacznie szybciej niż np. w rejestrze przesuwającym (Dlaczego?).



Rys. 18.13 Jeżeli $S=0$ - brak przesunięcia, gdy $S=1$ - przesunięcie o 4 bity w lewo

Mówiąc o przesunięciu musimy zrobić założenie o wartościach, które „wchodzą” na zwolnioną pozycję oraz ustalić co się dzieje z wartością pozycji „wysuwanej”. Przesunięcia mogą być cykliczne (*rotate*) lub logiczne (*shift*), czyli takie, w których na zwolnioną pozycję wpisywane są zera (wyróżnia się jeszcze przesunięcia arytmetyczne, w których zachowuje się bit znaku). Jeżeli powinniśmy przesunąć (cyklicznie) wartości 8-bitowego wejścia w lewo o 4 pozycje (bity), to możliwa jest realizacja przesuwnika jaką przedstawiliśmy na rys. 18.13.

Jeżeli poczynimy założenie, że stosunkowo łatwo zrealizujemy przesunięcie o m bitów, gdzie m jest potęgą dwójki, to możemy zrealizować przesunięcie o dowolną liczbę bitów. Na przykład, przesunięcie o 13 pozycji realizujemy dokonując kolejno przesunięcia o 1, 4 i 8 pozycji. Spostrzeżenie to, doprowadza nas do układu przesuwnika kaskadowego (*barrel shifter*), którego poglądową ideę działania podaliśmy na rys. 18.14.

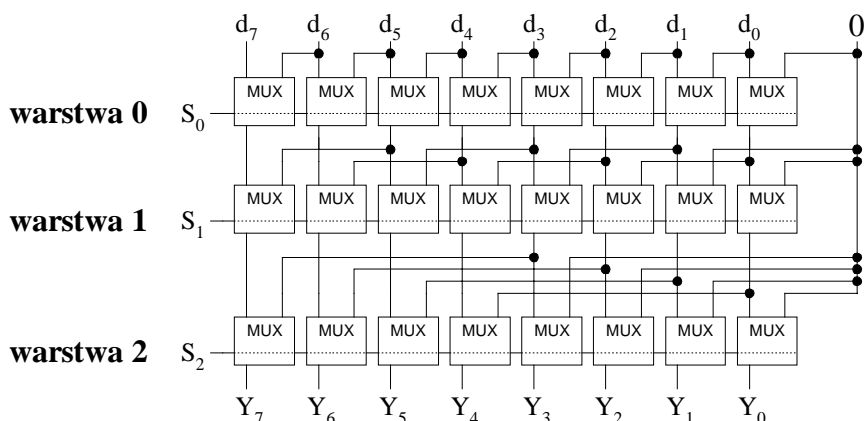
Układ realizuje przesunięcie logiczne w lewo (dlatego z prawej strony wsuwane są wartości 0, a wysuwane wartości są gubione).

Każda warstwa (kaskada) przesuwnika realizuje przesunięcie o 2^i pozycji, gdy $S_i=1$ albo przenosi wartości z wejścia na wyjście, gdy $S_i=0$, gdzie i jest numerem warstwy, $0 \leq i \leq 2$, czyli najwyżej położona warstwa przesuw o 1, kolejne niżej położone o 2 i 4. Cały przesuwnik maksymalnie przesuw o 7 pozycji. Jeżeli przykładowo, chcemy zrealizować przesunięcie o 5 pozycji w lewo, należy na wejścia sterujące podać następujące wartości: $S_0=1$, $S_1=0$ oraz $S_2=1$. Jeżeli przesuwamy o 6 pozycji, należy podać: $S_0=0$, $S_1=1$ oraz $S_2=1$.

Zauważmy, że wielkość o którą przesuwamy wystarczy przedstawić binarnie i podać na wejścia sterujące. Jest to o tyle istotne, że gdy przesuwnik jest częścią jednostki arytmetycznej

tyczno-logicznej komputera i ma być realizowany rozkaz przesunięcia, to wielkość o jaką przesuwamy kodowana jest binarnie, czyli wystarczy podać ją na wejścia sterujące przesuwnika.

Układ jest bardzo szybki, ponieważ realizuje przesunięcie w czasie równym trzech opóźnień pojedynczej kaskady. Jako ćwiczenie, proponujemy Czytelnikowi narysowanie przesuwnika kaskadowego przesuwającego zarówno w prawo jak i w lewo.



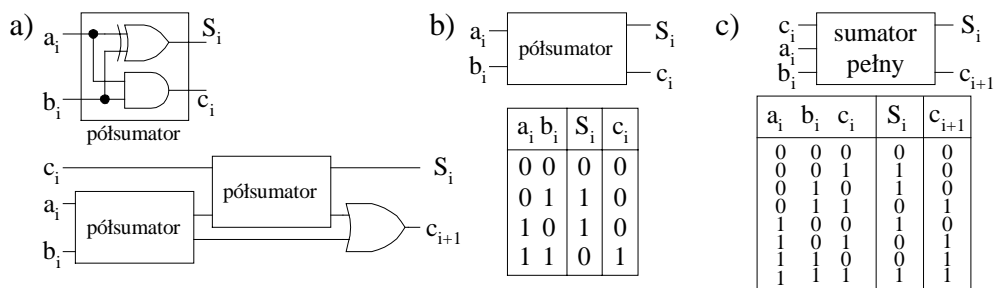
Rys. 18.14 Przesuwnik kaskadowy realizujący przesunięcie logiczne w lewo

18.5 Sumatory

Wiele zastosowań wymaga realizacji arytmetycznego dodawania wartości binarnych. Najprostszym układem dodającym dwie wartości jednobitowe jest półsumator (*half adder*). Wynikiem dodawania jest dwubitowa wartość z przedziału [0..2]. Na rys. 18.15 pokazaliśmy układ, oznaczenie oraz tablice prawdy półsumatora. Mniej znaczący bit wyniku oznaczyliśmy przez S_i (suma), a bardziej znaczący - c_i (przeniesienie).

Aby dodać dwie liczby wielobitowe, musimy uwzględnić przeniesienie pomiędzy sąsiednimi bitami. Sumator pełny (*full adder*) dodaje dwie wartości binarne oraz wartość przeniesienia wejściowego (*carry-in*) z poprzedniej pozycji. Wynikiem jest binarna wartość sumy oraz przeniesienia wyjściowego (*carry-out*). Na rys. 18.15 pokazaliśmy układ, oznaczenie oraz tablice prawdy sumatora pełnego.

Standardowy n-bitowy sumator z propagacją przeniesień (*ripple-carry adder*, *carry-propagate adder*) jest łańcuchem n sumatorów pełnych (Rys. 18.16).



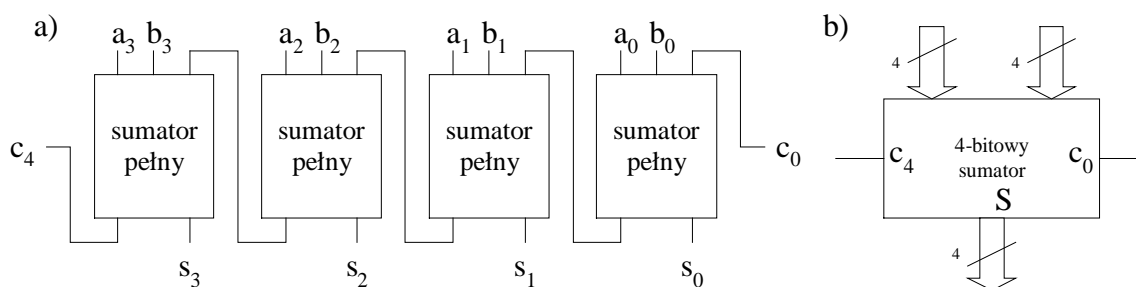
Rys. 18.15 Półsumator i sumator pełny, a - układy, b,c – oznaczenia i tablice prawdy

Sumator taki pozwala dodać dowolne dwa słowa binarne o długości n. Czas wytworzenia wyniku dodawania (po którym zostaną ustalone poprawne wartości s_3 i c_4) jest sumą czasów

propagacji wszystkich przeniesień c_i wytwarzanych w sumatorach pełnych, ponieważ w najgorszym przypadku przeniesienie propaguje się przez wszystkie sumatory.

Na przykład, gdy dodajemy 111...11 do 000...01, to przeniesienie propaguje się przez wszystkie sumatory (patrz też ćwiczenia). Z tego też względu są to układy dosyć wolne.

Znacznie lepszy (szybszy) sumator n -bitowy uzyskamy, jeżeli przewidzimy wartość przeniesienia wyjściowego dowolnego stopnia. Ideę pomysłu ilustruje rysunek 18.17, przy założeniu, że wartość sumy i -tego stopnia wyrazimy następująco: $s_i = a_i \oplus b_i \oplus c_i$.



Rys. 18.16 Czterobitowy sumator z propagacją przeniesień: a - układ, b - oznaczenie

Zauważmy, że układ przewidywania przeniesienia wytwarza c_i jako funkcję $a_0...a_{i-1}, b_0...b_{i-1}$ oraz c_0 . Zdefiniujmy teraz dwa ważne pojęcia. Mówimy, że i -ty stopień generuje przeniesienie, gdy wytworzone $c_{i+1}=1$ jest niezależne od $a_0...a_{i-1}, b_0...b_{i-1}$ oraz c_0 . Przyjmijmy także, że i -ty stopień propaguje przeniesienie, gdy $c_{i+1}=1$ przy $c_i=1$. Zgodnie z definicjami, możemy teraz napisać wyrażenia logiczne dla sygnału generacji przeniesienia g_i oraz propagacji przeniesienia p_i , dla każdego stopnia sumatora:

$$g_i = a_i b_i$$

$$p_i = a_i \oplus b_i$$

Innymi słowy, i -ty stopień generuje przeniesienie, gdy bity obu składników są równe 1 oraz propaguje przeniesienie, gdy przynajmniej jeden z nich jest równy 1. Przeniesienie wyjściowe z i -tego stopnia może być wyrażone jako funkcja sygnału generacji i propagacji w sposób następujący: $c_{i+1} = g_i + p_i c_i$. Czyli

$$c_1 = g_0 + p_0 c_0$$

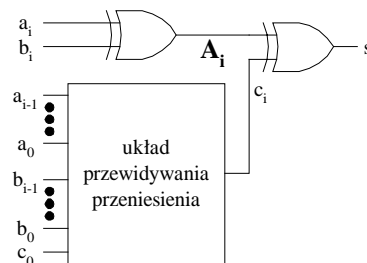
$$c_2 = g_1 + p_1 c_1 = g_1 + p_1 (g_0 + p_0 c_0) = g_1 + p_1 g_0 + p_1 p_0 c_0$$

$$c_3 = g_2 + p_2 c_2 = g_2 + p_2 (g_1 + p_1 g_0 + p_1 p_0 c_0) = g_2 + p_2 g_1 + p_2 p_1 g_0 + p_2 p_1 p_0 c_0$$

$$c_4 = g_3 + p_3 c_3 = g_3 + p_3 (g_2 + p_2 g_1 + p_2 p_1 g_0 + p_2 p_1 p_0 c_0) = g_3 + p_3 g_2 + p_3 p_2 g_1 + p_3 p_2 p_1 g_0 + p_3 p_2 p_1 p_0 c_0$$

Każde z powyższych wyrażeń odpowiada układowi trójpoziomowemu ze względu na wnoszone opóźnienie - pierwszy poziom wytwarza sygnały generowania i przenoszenia przeniesień, a drugi i trzeci, to realizacja powyższych sum iloczynów. Sumator, w którym wszystkie przeniesienia wyznaczane są jednocześnie (przewidując) na podstawie funkcji generacji i propagacji przeniesień, nazywany jest sumatorem z przewidywaniem przeniesienia (*carry lookahead adder*).

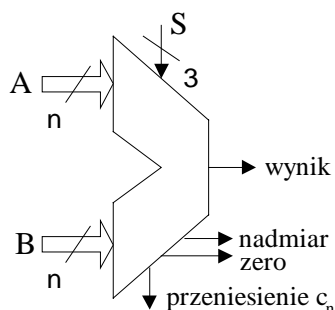
Istnieje wiele innych propozycji przyspieszenia operacji sumowania, ale wykracza to poza ramy tego rozdziału [5].



Rys. 18.17 Przewidywanie przeniesienia

18.6 Jednostka arytmetyczno-logiczna

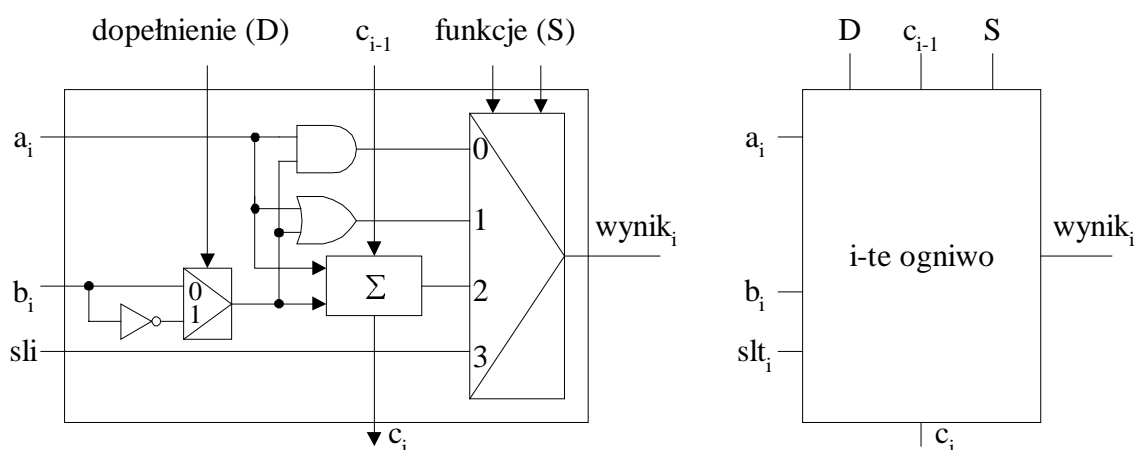
Układ kombinacyjny realizujący operacje arytmetyczne (dodaj, odejmij) nazywamy jednostką arytmetyczną. Zazwyczaj jednostki te realizują także operacje logiczne (AND, OR, NOT), dlatego układy takie nazywamy jednostkami arytmetyczno-logicznymi (*arithmetic and logic units, ALU*). ALU to "mięśnie" każdego komputera.



Rys. 18.18 Ogólny schemat ALU

Spróbujemy zaprojektować uproszczoną jednostkę przy założeniu, że Czytelnik zaznajomiony jest z reprezentacją liczb binarnych w kodzie uzupełnieniowym do dwóch (*two's complement*) - w skrócie U2. Nazwa kodu pochodzi z własności tej reprezentacji liczb, ponieważ suma n-bitowych liczb x i $-x$ jest równa 2^n (binarnie 100...00). Stąd cenna własność kodu - odejmowanie realizujemy poprzez dodanie uzupełnionego do dwóch odjemnika (uzupełnienie to otrzymujemy przez dopełnienie każdego bitu odjemnika i dodaniu 1 do najmniej znaczącej pozycji), co z punktu widzenia realizacji sprowadza się do wykorzystania jednego sumatora n-bitowego (zauważmy, że operacja uzupełniania też wykorzystuje dodawanie).

Jeżeli umówimy się, że waga pozycji najbardziej znaczącej wynosi nie 2^{n-1} , ale -2^{n-1} , to wszystkie liczby zaczynające się od 0 będą dodatnie, a liczby zaczynające się od 1 będą ujemne (czyli najbardziej znaczący bit jest bitem znaku).



Rys. 18.19 1-bitowe ogniwo ALU realizujące funkcje and, or, add, sub i slt

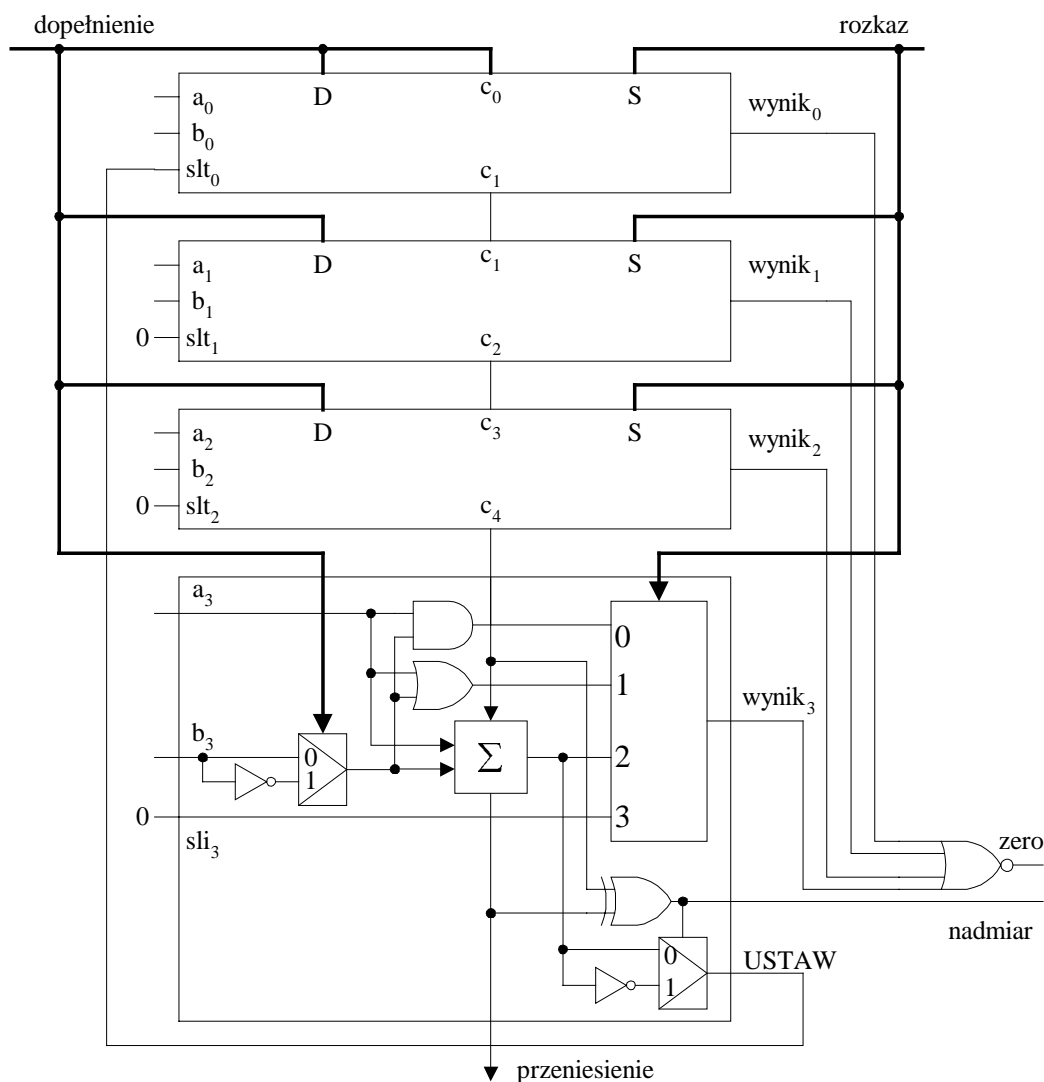
Komputer wykonuje wszystkie operacje na liczbach (słowach) o ustalonej długości (zazwyczaj $n=32$), co jest źródłem komplikacji przy wykonywaniu operacji arytmetycznych. Jeżeli dwie liczby dają się zakodować w U2 z wykorzystaniem n bitów, to może się zdarzyć, że albo sumy, albo różnicy nie można w tak ograniczonym słowie zapisać. Na określenie takiej sytuacji używamy słowa nadmiar (*overflow*). Ponieważ wynik operacji jest błędny, więc ALU powinna sygnalizować wystąpienie nadmiaru. Warunki wystąpienia nadmiaru są w różnych kodach liczbowych różne - dla kodu U2 warunek ten jest następujący:

$$V = c_n \oplus c_{n-1},$$

czyli nadmiar wystąpi wtedy, gdy przeniesienia z dwóch najbardziej znaczących pozycji są różne (pozostawiamy Czytelnikowi udowodnienie tego warunku).

Założmy, że projektowana tutaj jednostka arytmetyczno-logiczna powinna realizować następujące funkcje: arytmetyczne dodawanie (rozkaz add), odejmowanie (rozkaz sub), logiczne dodawanie (rozkaz or) i mnożenie (rozkaz and) a także funkcję "ustaw gdy mniejsze niż" (rozkaz slt). ALU powinna sygnalizować nadmiar oraz zerowy wynik operacji.

W zależności od aktualnie wykonywanego rozkazu (dla uproszczenia zdefiniowaliśmy ich tylko pięć), ALU powinna realizować różne funkcje. Aby to umożliwić, oprócz dwóch n-bitowych wejść, poprzez które przekazujemy operandy rozkazów, ALU powinna posiadać wejścia sterujące, informujące o funkcji do realizacji. Ponieważ zdefiniowaliśmy 5 rozkazów, więc wystarczą 3 linie sterujące. Na rys. 18.18 przedstawiliśmy ogólny schemat ALU.



Rys. 18.20 Końcowa wersja 4-bitowego ALU

Konstruowanie ALU rozpoczniemy od zaprojektowania pojedynczego 1-bitowego ogniwa (rys. 18.19). Do realizacji funkcji logicznych and i or wykorzystamy bramki logiczne AND i OR a do operacji arytmetycznych pojedyncze ogniwo sumatora.

Cztery pojedyncze ogniwa ALU połączymy tak jak to przedstawiliśmy na rys. 18.20. By wykonać operację $a \text{ AND } b$ ($a \text{ OR } b$) należy ustalić: $D=0$, $c_0=0$ oraz $S=00(01)$. Dodawanie (odejmowanie) wykonamy, gdy: $D=0(1)$, $c_0=0(1)$, $S=10$.

Zauważmy, że ostatnie ogniwo posiada dwa dodatkowe wyjścia, których nie ma na rys. 18.19 (realizuje dodatkowe funkcje związane z rozkazem slt oraz sygnalizacją nadmiaru).

Wiemy już, że nadmiar wystąpi wtedy, gdy przeniesienia z dwóch najbardziej znaczących pozycji są różne, dlatego układ sygnalizujący nadmiar umieściliśmy w najbardziej znaczącym ogniwie, ponieważ oba interesujące nas przeniesienia są tam dostępne. Do realizacji warunku wystarczy wykorzystać jedną bramkę EOR.

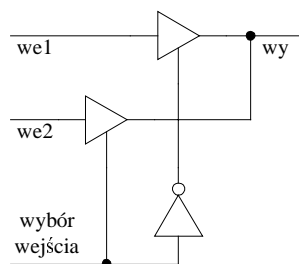
Sygnał USTAW związany jest z realizacją rozkazu slt, któremu w tym miejscu poświęcimy więcej uwagi. Rozkaz ten występuje na liście rozkazów procesora MIPS.

Jeżeli wykonujemy rozkaz slt r2,r3,r4, (przez rx oznaczyliśmy rejestry procesora), to procesor MIPS przy spełnieniu warunku $[r2] < [r3]$ powinien do rejestru r4 wstawić wartość 1 (0 w każdym innym przypadku, [rx] - zawartość rejestru). Zauważmy, że gdy $A < B$ to $A - B < 0$ i bit znaku jest równy 1. Wystarczy zatem wartość bitu znaku przekazać do wejścia slt_0 , a do pozostałych ogniw $slt_i = 0$, by spełnić specyfikację rozkazu.

Jednak powyższa realizacja nie jest poprawna, ponieważ w sytuacji wystąpienia nadmiaru bit znaku jest błędny (co łatwo sprawdzić np. dla $n=4$, $A = -8_{10}$ (1000), $B = 7_{10}$ (0111), $A - B = 0001$, co sugeruje, że A nie jest mniejsze od B, a tak nie jest). Dlatego, przy nadmiarze należy dopełnić wartość bitu znaku, przed wysłaniem go na wyjście USTAW. Na rys. 18.20. powyższą zależność zrealizowaliśmy za pomocą MUX.

18.7 Elementy trójstanowe

Dotychczas rozważane elementy logiczne posiadały wyjście dwustanowe, odpowiadające logicznym wartościom 0 i 1. Produkowane są jednak elementy, posiadające trzeci stan "elektryczny", nazywany stanem wysokiej impedancji, który nie jest stanem logicznym. Mówiąc najprościej, element którego wyjście jest wprowadzone w stan wysokiej impedancji, zachowuje się podobnie do sytuacji odłączenia danego wyjścia od układu, do którego galwanicznie nadal jest podłączony. Elementy trójstanowe posiadają dodatkowe wejście sterujące, pozwalające wprowadzić element w stan wysokiej impedancji.



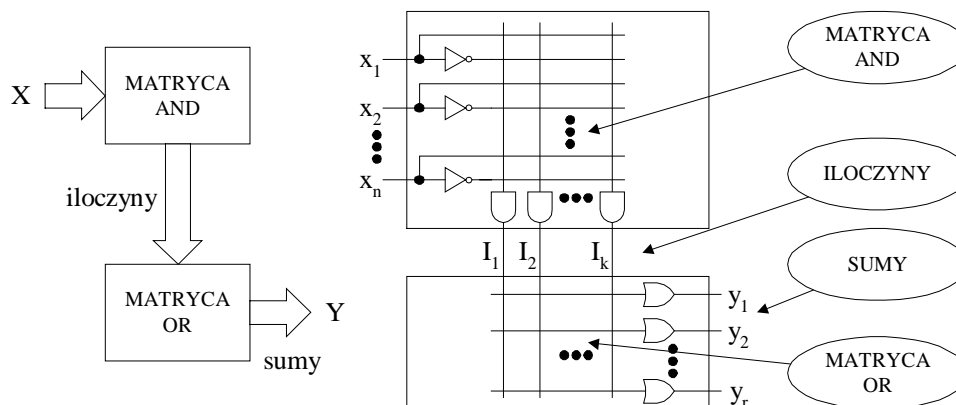
Rys. 18.21 Wykorzystanie buforów trójstanowych

Na rys. 18.21 przedstawiliśmy ilustrację wykorzystania trójstanowych buforów. Działanie układu jest następujące – gdy sygnał „wybór wejścia” przyjmuje wartość logiczną 0(1) na wyjście układu przekazywana jest wartość wejścia we1(we2). Oznacza to, że przy podaniu na wejście sterujące wartości logicznej 1, bufor przekazuje bez zmian wartość wejścia na swoje wyjście. W przeciwnym przypadku bufor jest wprowadzony w stan wysokiej impedancji.

Elementy z wyjściami trójstanowymi są wykorzystywane przy bezpośrednim podłączeniu różnych urządzeń (nadajników informacji) do magistrali (jest to wiązka przewodów, przez które przesyłane są dane binarne). Wystarczy jedynie odpowiednio sterować wejściami ustawiającymi wyjścia trójstanowe nadajnika, aby zapobiec ewentualnym konfliktom dostępu do wspólnej magistrali. Unikniemy konfliktu, jeżeli w danej chwili będziemy przysyłać dane tylko z jednego nadajnika - pozostałe nadajniki podpięte do magistrali powinny być ustawione w stan wysokiej impedancji. Z tego względu magistrala wymaga istnienia układu arbitrażowego, rozstrzygającego konflikty i przyznającego prawo nadawania.

18.8 Układy programowalne

Programowalne układy logiczne (*PLD, Programmable Logic Devices*) wytwarzane są w dużej skali scalenia jako "regularne" struktury, które konstruktor przystosowuje do własnych potrzeb zastosowaniowych. Przystosowanie, o którym mowa, nazywamy także programowaniem układu.



Rys. 18.22 Ogólna struktura PLD

Ogólną strukturę układów PLD przedstawiliśmy na rys. 18.22. Składa się ona z dwóch matryc - matrycy AND oraz matrycy OR.

W zależności od tego, którą z matryc możemy zaprogramować, wyróżnia się różne typy struktur PLD (tabela 18.1).

Tabela 18.1

Matryca	Typ struktury PLD		
	PLA	PAL	PROM (PLE)
AND	programowalna	programowalna	nieprogramowalna
OR	programowalna	nieprogramowalna	programowalna

PLA - Programmable Logic Array PAL - Programmable Array Logic
PROM - Programmable Read-Only Memory PLE - Programmable Logic Element

Jak wynika z tabeli 18.1 struktura PLA pozwala programować obie matryce. Zazwyczaj liczba iloczynów jest znacznie mniejsza od 2^n , czyli możemy utworzyć tylko niektóre iloczyny. Z matrycy OR możemy utworzyć r sum złożonych z dowolnych iloczynów utworzonych w matrycy AND.

Przykład:

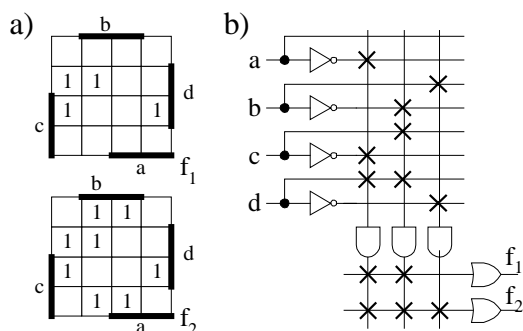
Wymiary produkowanych PLA są duże (np. $k=96$, $r=8$, $n=16$), dlatego poniższy przykład syntezy zespołu dwóch funkcji przełączających ma jedynie charakter dydaktyczny. Niech będą dane funkcje f_1 oraz f_2 przedstawione na rys. 18.23a. Po minimalizacji zespołu dwóch funkcji możemy napisać:

$$f_1(abcd) = \bar{a} \cdot \bar{c} \cdot d + \bar{b} \cdot c \cdot d$$

$$f_2(abcd) = \bar{a} \cdot \bar{c} \cdot d + \bar{b} \cdot c \cdot d + b \cdot \bar{d}$$

Na rys. 18.23b przedstawiliśmy realizację układową. Krzyżykiem zaznaczyliśmy punkty programujące, czyli wskazaliśmy które zmienne wejściowe (afirmowane albo dopełnione) wchodzi do iloczynu wytwarzanego na linii iloczynowej oraz na liniach poziomych prowadzących do "bramek" OR wskazaliśmy iloczyny wchodzące do sumy logicznej. W matrycy AND utworzyliśmy trzy iloczyny: $\bar{a} \cdot \bar{c} \cdot d$, $\bar{b} \cdot c \cdot d$, $b \cdot \bar{d}$ a w matrycy OR dwie sumy f_1 oraz f_2 . Zauważmy, że dwa iloczyny są wspólne dla obu funkcji. Zaletą struktury PLA jest jej duża

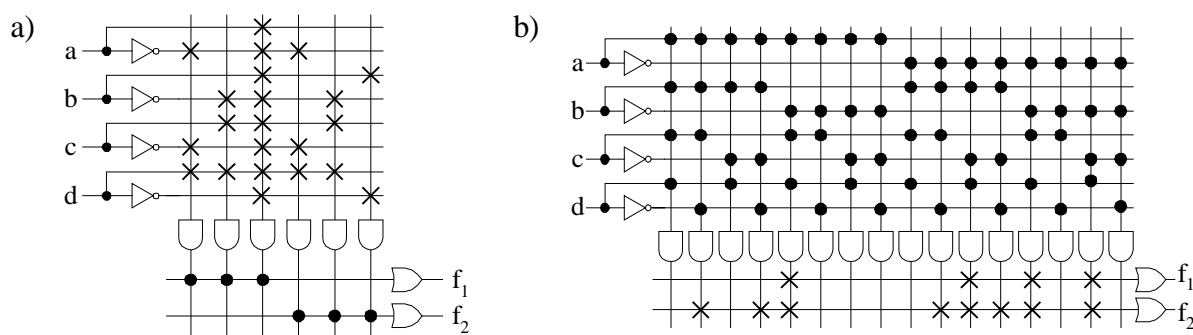
elastyczność polegająca na tym, że możemy programować obie matryce. Jednak w produkowanych strukturach PLA, w stosunku do liczby zmiennych wejściowych, pewnym ograniczeniem jest liczba dostępnych linii iloczynowych, co zmusza projektanta do stosowania algorytmów minimalizacji zespołu funkcji przełączających.



Rys. 18.23 Realizacja układu dla dwóch funkcji na strukturze PLA, a - tablice Karnaugh, b -

Realizację układową tych samych dwóch funkcji (rys. 18.23a) na strukturze PAL przedstawił na rys. 18.24a. Założyliśmy, że każde wyjście nieprogramowalnej matrycy OR jest na stałe podłączone do trzech linii iloczynowych (dla odróżnienia, zaznaczyliśmy je kropką). Zauważmy, że obie funkcje zostały zminimalizowane osobno, ponieważ wspólne iloczyny dla obu funkcji muszą być teraz realizowane oddzielnie. Trzecia linia iloczynowa przekazuje na wyjście wartość 0.

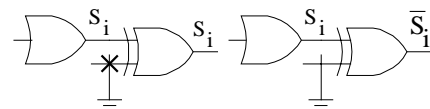
Trzecią strukturą wymienioną w tabeli 18.1 jest struktura PLE (nowsza realizacja pamięci PROM). Liczba linii iloczynowych jest równa $k=2^n$, czyli wytwarzane są wszystkie iloczyny zupełne. Matryca AND nie jest programowalna - co zaznaczyliśmy dużą kropką (rys. 18.24b). Matrycę OR można programować wybierając dowolne iloczyny. Struktura ta pozwala zaprogramować tyle różnych funkcji ile jest wyjść z matrycy OR. Realizowane funkcje nie muszą być minimalizowane (realizujemy je wprost z postaci kanonicznej bądź tablicy prawdy).



Rys. 18.24 Realizacja w strukturze: a - PAL, b - PLE (PROM)

Podsumowując powyższe trzy realizacje układowe należy stwierdzić, że efektywne wykorzystanie PLA wymaga minimalizacji zespołu funkcji, PAL minimalizacji każdej funkcji z osobna a PLE (PROM) nie wymaga żadnego procesu minimalizacji. Popularność wykorzystania PLA spowodowała opracowanie specjalnych algorytmów minimalizacyjnych, z których najbardziej znany jest ESPRESSO [3].

Standardowe struktury programowalne wyposażone są w dodatkowe udogodnienia w postaci możliwości programowania polaryzacji wyjść (czy wyjście ma być afirmowane czy dopełnione), sprzężeń zwrotnych czy dwukierunkowych wyprowadzeń.



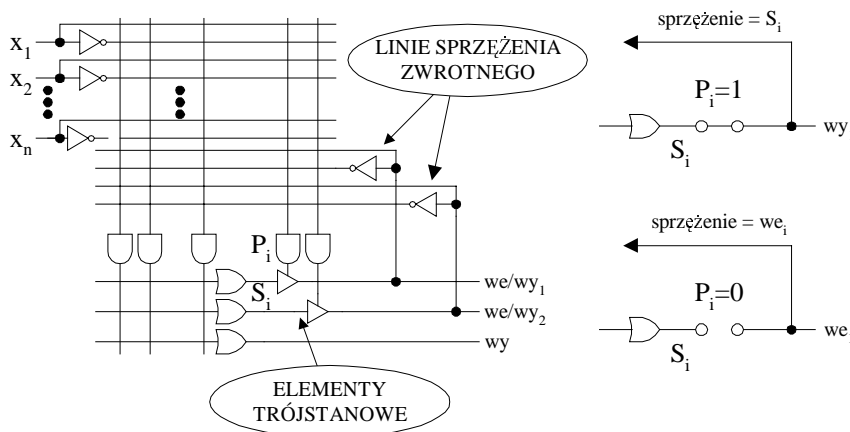
Rys. 18.25 Programowanie polaryzacji.

Na rys. 18.25 przedstawiliśmy programowanie polaryzacji wyjść z matrycy OR. Proponujemy Czytelnikowi wykorzystanie tej możliwości do realizacji postaci dysjunkcyjnej i koniunkcyjnej zadanej funkcji.

Na rys. 18.26 pokazaliśmy ilustrację działania dwukierunkowych wyprowadzeń, które przede wszystkim wykorzystujemy do zwiększenia liczby wejść struktury programowalnej.

W tabeli 18.2 podaliśmy wybrane elementy PAL, PLA oraz PROM

Więcej o urządzeniach programowalnych możemy dowiedzieć się z literatury [2].



Rys. 18.26 Ilustracja działania dwukierunkowych wyprowadzeń

Tabela 18.2

Układ	Struktura	Liczba wejść	Liczba iloczynów	Liczba wyjść	Polaryzacja wyjścia
PAL16L8	PAL	16	8	8	Poziom niski
PAL14H4	PAL	14	4	4	Poziom wysoki
PAL16C1	PAL	16	16	1	Komplementarna
PAL18P8	PAL	18	8	8	Programowalna
PLS100	PLA	16	48	8	Programowalna
PLS153	PLA	16	42	10	Programowalna
82S129	PROM	5	32	8	Poziom wysoki
82S135	PROM	8	256	8	Poziom wysoki
82S147	PROM	9	512	8	Poziom wysoki
82S191	PROM	11	2048	8	Poziom wysoki

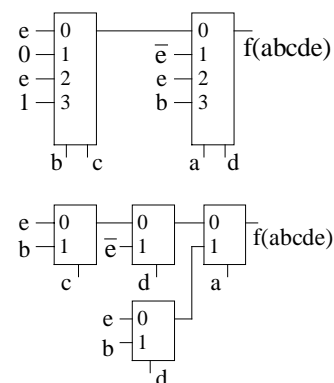
Literatura

- [1] Molski M., Modułowe i mikroprogramowalne układy cyfrowe, WKiŁ, Warszawa, 1986,
- [2] Łuba T., Jasiński K., Zbierzchowski B., Specjalizowane układy cyfrowe w strukturach PLD i FPGA, WKiŁ, Warszawa, 1997.
- [3] Brayton R.K., Hachtel G.C., McMullen C.T., Sangiovanni-Vincentelli A.L., Logic Minimization Algorithms for VLSI Synthesis, Boston, Kluwer Academic Publishers, 1984.
- [4] Hurst S.L., Miller D.M., Muzio J.C., Spectral Techniques in Digital Logic, Academic Press, 1985
- [5] Biernat J., Arytmetyka komputerów, PWN Warszawa 1996.

ĆWICZENIA

1. Pokaż, że jeżeli multiplekser ma k linii sterujących (zatem powinien posiadać 2^k linii informacyjnych) to można go wykorzystać do realizacji dowolnej funkcji $(k+1)$ zmiennych.
2. Po wykazaniu, że układy multiplekserowe z rys. 18.25 realizują tę samą funkcję $f(abcde) = \bar{a}\bar{d}\bar{e} + abd + \bar{a}bcd + \bar{c}\bar{d}e + a\bar{d}e$, zastanów się jak można te układy zsyntezować[4].

3. Zakładając, że oba składniki sumatora z propagacją przeniesień są postaci 11...11 pokazać, że opóźnienie wprowadzane przez sumator jest równe: $t_{sum} = t_{abc0} + (n-2)t_{c0ci} + t_{c0s}$, gdzie t_{abc0} - opóźnienie od wejść a,b do c0 w najmniej znaczącym stopniu sumatora, t_{c0ci} - opóźnienie od wejścia c0 do ci w stopniach środkowych, t_{c0s} - opóźnienie od c0 do wyjścia S.
4. Zaproponuj układ wykrywający zanik propagacji przeniesienia w sumatorze z propagacją przeniesień (jeżeli wynik jest ustalony, to można rozpocząć następną operację dodawania).
5. Narysuj pełny układ czterobitowego sumatora z przewidywaniem przeniesień (wykorzystaj bramki NOR, NAND oraz XOR).
6. Na rys. 18.16 pewną linię oznaczono literą A. Czy można pokazać, że $A_i = p_i \cdot \bar{g}_i$.
7. Narysuj pełny układ czterobitowego sumatora z przewidywaniem przeniesień.
8. Zrealizuj sumator z propagacją przeniesień dodający dwie dwubitowe wartości binarne, wykorzystując do tego celu programowalne urządzenie z dwukierunkowymi wyjściami. Przyjmijmy, że urządzenie programowalne ma cztery wejścia dedykowane, trzy wyjścia dedykowane oraz dwa wyjścia dwukierunkowe.
9. Dlaczego na rys. 18.19 sygnał dopełnienie połączono z wejściem c₀?
10. Na rys. 18.19 dwie linie sygnałowe zaznaczono pogrubioną linią. Sygnał rozkaz steruje multiplekserami czterowejściowymi, dlatego musi reprezentować wartości 2-bitowe. Sygnał dopełnienie jest 1-bitowy. Oznacza to, że sygnały sterujące pracą ALU są 3-bitowe. Należy przyporządkować 3-bitowe wartości do pięciu rozkazów (and, or, add, sub i slt) tak by ALU z rys. 18.19 poprawnie je realizowało.
11. Przesuwnik zrealizowany multiplekserach (rys. 18.13) przesuwają 8-bitową wielkość w lewo, wstawiając wartość zero z prawej strony. Co należy zrobić, by przesuwnik wykonywał przesunięcie rotacyjne?
12. Zsyntezuj przesuwnik 8-mio bitowych wielkości. Wykorzystaj do realizacji układu:
 - a) czterowejściowe i ośmiowejściowe bramki. Policz liczbę bramek. Wybierz odpowiednie układy MSI zawierające wymagane bramki. Czy potrzebujesz sześćdziesięciu czterech czterowejściowych oraz ośmiu ośmiowejściowych bramek? Czy wybrałeś czterdzieści pakietów układów MSI?
 - b) Ośmiowejściowe multipleksery. Czy wybrałeś osiem układów MSI?
 - c) ROM (lub PLA/PAL). Ile wybrałeś układów?
13. Rozwiązanie zadania 12 wskazuje, że realizacja przesuwnika z wykorzystaniem bramek, multiplekserów czy ROM nie jest satysfakcjonująca ze względu na liczbę wykorzystywanych elementów (w szczególności bramek). Poszukaj w literaturze przedmiotu rozwiązania przesuwnika (*barrel shifter*), do którego realizacji nie użyto bramek, multiplekserów czy ROM/PLA.



Rys. 18.27 Różne realizacje tej samej funkcji