KATEDRA AUTOMATYKI, BIOMECHANIKI
I MECHATRONIKI

Opracowanie teoretyczne

TEMAT:

Teoretyczne podstawy syntezy układów logicznych

Aktualizacja: luty 2022
Opracowanie: dr inż. Grzegorz Wasilewski, dr inż. Adam Wijata,
dr inż. Krystian Polczyński, mgr inż. Mateusz Wojna
wersja: 1.1

Spis treści

1 Wstęp

Od wielu lat układy logiczne znajdują w technice coraz szersze zastosowanie. W oparciu o układy logiczne rozwinęły się takie dziedziny techniki jak:

- elektroniczna technika obliczeniowa,
- teletechnika,
- miernictwo elektryczne,
- elektronika,
- technika sterowania procesami przemysłowymi i badawczymi oraz szereg pokrewnych dziedzin, wykorzystujących zasadniczą cechę układów logicznych jaką jest istnienie dwóch stanów przesyłanego sygnału.

Założenie dwóch wartości (stan 0 i stan 1) nominalnych sygnału decyduje o projektowaniu, konstrukcji i opisie układu logicznego (zwanego także układem przełączającym).

Układy logiczne mogą być realizowane:

- elektronicznie,
- elektromechanicznie,
- pneumatycznie.

Stan 1 jest określony istnieniem napięcia elektrycznego lub ciśnienia na danym poziomie, np.: +5V, 2.5 bara.

Stan 0 z kolei jest określony najczęściej poziomem sygnału bliskim zera, np. 0.25V, 0.05 bara lub ewentualnie równym 0.

Operacjami matematycznymi na zmiennych binarnych przyjmujących tylko dwie wartości 0 lub 1 (prawda lub fałsz) zajmują się algebry logiki, zwłaszcza zerojedynkowa (dwuelementowa) algebra Boole'a, a funkcje logiczne realizowane przez te elementy nazywane są funkcjami boolowskimi lub zerojedynkowymi. Ich zrozumienie stanowi podstawę projektowania układów przełączających w automatyce.

Operacjami logicznymi, które można wykonywać na zmiennych binarnych są:

- suma logiczna (LUB, OR),
- iloczyn logiczny (I, AND),
- negacja (NIE, NOT).

Tabela 1.1 Oznaczenia operacji logicznych

	logika	**C++/Java**	**technika cyfrowa**	**MS Excel**
suma logiczna (*LUB, OR*)	$a \vee b$	$a \mid\mid b$	$a + b$	$\mathrm{LUB}(a; b)$
iloczyn logiczny (*I, AND*)	$a \wedge b$	$a \,\&\&\, b$	$a \cdot b$	$\mathrm{ORAZ}(a; b)$
negacja (*NIE, NOT*)	$\neg a$	$!\, a$	$\bar{a}$	$\mathrm{NIE}(a)$

2 Podstawowe operacje i zasady rachunku Boole'a

Suma logiczna dwóch zmiennych:

a	b	$a+b$
0	0	0
0	1	1
1	0	1
1	1	1

Ogólnie suma logiczna jest równa 0 wtedy i tylko wtedy, jeżeli wszystkie składniki sumy są równe 0. Wynikają stąd własności:

$$a + 0 = a,$$

$$a + 1 = 1,$$

gdzie a – zmienna o dowolnej wartości (0 lub 1).

Iloczyn logiczny dwóch zmiennych:

a	b	$a \cdot b$
0	0	0
0	1	0
1	0	0
1	1	1

Ogólnie iloczyn logiczny wynosi 1 tylko wtedym jeżeli wszystkie czynniki są równe 1; stąd wynikają własności:

$$a \cdot 0 = 0,$$

$$a \cdot 1 = a,$$

gdzie zmienna a może przyjmować dowolną wartość.

Negacja:

a	$\bar{a}$
0	1
1	0

Ogólnie

$$\bar{a} \neq a,$$

natomiast podwójna negacja

$$\bar{\bar{a}} = a.$$

Łatwo wykazać związki:

$$\bar{a} \cdot a = 0,$$

$$\bar{a} + a = 1.$$

Dla rachunku zerojedynkowego obowiązują następujące reguły:

1) prawo łączności iloczynu

$$a \cdot b \cdot c = (a \cdot b) \cdot c = a \cdot (b \cdot c), \tag{2.1}$$

2) prawo łączności sumy

$$a + b + c = (a + b) + c = a + (b + c), \tag{2.2}$$

3) prawo przemienności iloczynu

$$a \cdot b = b \cdot a, \tag{2.3}$$

4) prawo przemienności sumy

$$a + b = b + a, \tag{2.4}$$

5) prawo powtórzenia iloczynu

$$a \cdot a = a, \tag{2.5}$$

6) prawo powtórzenia sumy

$$a + a = a, \tag{2.6}$$

7) prawo rozdzielności iloczynu względem sumy

$$a \cdot (b + c) = (a \cdot b) + (a \cdot c), \tag{2.7}$$

8) prawo rozdzielności sumy względem iloczynu

$$a + (b \cdot c) = (a + b) \cdot (a + c). \tag{2.8}$$

Korzystając z prawa 7) można udowodnić:

$$ab + a\bar{b} = a\left(b + \bar{b}\right) = a \cdot 1 = a. \tag{2.9}$$

Korzystając z reguły 8) można wykazać, że:

$$(a + b)\left(a + \bar{b}\right) = a + \left(b \cdot \bar{b}\right) = a + 0 = a, \tag{2.10}$$

$$a + b\bar{a} = (a + b)(a + \bar{a}) = (a + b) \cdot 1 = a + b. \tag{2.11}$$

Powyższe związki są nazywane „regułami sklejania".

Zasadnicze znaczenie w rachunku Boole'owskim mają prawa de Morgana dla dwóch zmiennych:

$$\overline{a \cdot b} = \bar{a} + \bar{b}, \tag{2.12}$$

$$\overline{a + b} = \bar{a} \cdot \bar{b}. \tag{2.13}$$

Prawa de Morgana są ważne również dla wielu zmiennych:

$$\overline{a_1 \cdot a_2 \cdot a_3 \cdot \ldots \cdot a_n} = \overline{a_1} + \overline{a_2} + \overline{a_3} + \cdots + \overline{a_n}, \tag{2.14}$$

$$\overline{a_1 + a_2 + a_3 + \cdots + a_n} = \overline{a_1} \cdot \overline{a_2} \cdot \overline{a_3} \cdot \ldots \cdot \overline{a_n}. \tag{2.15}$$

Stosując w powyższych prawach negację obu stron równania otrzymamy:

$$\overline{\overline{a_1} + \overline{a_2} + \overline{a_3} + \cdots + \overline{a_n}} = a_1 \cdot a_2 \cdot a_3 \cdot \ldots \cdot a_n, \tag{2.16}$$

$$\overline{\overline{a_1} \cdot \overline{a_2} \cdot \overline{a_3} \cdot \ldots \cdot \overline{a_n}} = a_1 + a_2 + a_3 + \cdots + a_n. \tag{2.17}$$

Widać stąd, że dowolny wzór można przedstawić za pomocą sumy logicznej i negacji albo iloczynu logicznego i negacji. Powyższy wniosek może służyć jako uzasadnienie faktu, że funkcja NOR (z ang. 'Not or', czyli „Nie lub", czyli zaprzeczenie alternatywy, tzn. $\overline{x+y}$) stanowi tzw. system funkcjonalnie pełny. Podobnie funkcja NAND (z ang. 'Not and', czyli 'Nie i', czyli zaprzeczenie koniunkcji, tzn. $\overline{xy}$) również spełnia kryterium systemu funkcjonalnie pełnego.

2.1 Przykład 1

Przedstawić w postaci iloczynowej funkcję $f = a + b + a \cdot c$.

$$\begin{aligned} f = a + b + a \cdot c = (a + b) + a \cdot c = \\ = (a + b + a)(a + b + c) = \\ = \left(\overline{\overline{a + b}}\right)\left(\overline{\overline{a + b + c}}\right) = \left(\overline{\bar{a} \cdot \bar{b}}\right)\left(\overline{\bar{a} \cdot \bar{b} \cdot \bar{c}}\right) \end{aligned} \qquad 2.18$$

2.2 Przykład 2

Przedstawić w postaci sumy funkcję $f = (ab + bc)a$.

$$\begin{aligned} f = (ab + bc)a = ab \cdot a + bc \cdot a = ab + abc = \overline{\overline{ab}} + \overline{\overline{abc}} = \\ = \left(\overline{\bar{a} + \bar{b}}\right) + \left(\overline{\bar{a} + \bar{b} + \bar{c}}\right) \end{aligned} \qquad 2.19$$

3 Synteza układów logicznych

W technice cyfrowej do algebraicznego zapisu dowolnych funkcji logicznych używa się symboli funkcji podstawowych, czyli znaku " + " dla alternatywy (sumy logicznej) i znaku " · " (razy) dla koniunkcji (iloczynu logicznego) oraz znaku " ¯ " (nie) nad symbolem zmiennej lub funkcji kilku zmiennych dla funkcji negacji (zaprzeczenia logicznego).

Znane są 2 metody zapisu funkcji:

- metoda postaci kanonicznej alternatywnej (składa się z elementarnych koniunkcji),
- metoda postaci kanonicznej koniunkcyjnej (składa się z elementarnych alternatyw).

Omówimy je na poniższym przykładzie.

3.1 Przykład 3

Dana jest funkcja $y = f(a, b, c)$ w postaci poniższej tabeli stanów.

a	b	c	y
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

Zapis funkcji w postaci kanonicznej alternatywnej przedstawia się następująco

$$y = \bar{a} \cdot \bar{b} \cdot \bar{c} + \bar{a} \cdot \bar{b} \cdot c + a \cdot \bar{b} \cdot \bar{c} + a \cdot \bar{b} \cdot c + a \cdot b \cdot \bar{c} + a \cdot b \cdot c. \qquad 3.1$$

Jak widać poszczególne składniki alternatywy są iloczynami logicznymi zmiennych a, b, c (lub ich negacji) ułożonych na podstawie tabeli stanów dla wartości funkcji $y = 1$ w taki sposób, że jeżeli wartość którejś zmiennej w tabeli jest 0, to w iloczynie logicznym odpowiada mu zapis zmiennej zanegowanej, natomiast jeśli w tabeli wartość zmiennej jest równa 1, to w iloczynie logicznym jest po prostu zapis tej zmiennej. Każdy iloczyn logiczny ma w ten sposób wartość logiczną równą 1, ponieważ przedstawia on sobą wyrażenie typu $1 \cdot 1 \cdot 1$.

Zapis funkcji w postaci kanonicznej koniunktywnej jest następujący

$$y = (a + \bar{b} + c) \cdot (a + \bar{b} + \bar{c}). \qquad 3.2$$

Zapis ten powstaje jako koniunkcja sum logicznych. Każda suma logiczna w tym zapisie odpowiada stanowi 0 funkcji y w tabeli stanów i powstaje w sposób taki, że jeżeli w tabeli stanów zmienna przyjmuje wartość 0, to w sumie logicznej odpowiada jej zapis zmiennej, natomiast jeśli posiada wartość 1, to należy w sumie logicznej ją zanegować. Każda suma logiczna ma w ten sposób postać $(0 + 0 + 0)$.

Zapisy funkcji 3.1 oraz 3.2 są tożsamościowo równe, co można udowodnić skracając zapis do minimalnego.

Z funkcji 3.1 wynika

$$\begin{aligned} y = \bar{a} \cdot \bar{b} \cdot \bar{c} + \bar{a} \cdot \bar{b} \cdot c + a \cdot \bar{b} \cdot \bar{c} + a \cdot \bar{b} \cdot c + a \cdot b \cdot \bar{c} + a \cdot b \cdot c \\ = (\bar{a}\bar{b}\bar{c} + \bar{a}\bar{b}c) + (a\bar{b}\bar{c} + a\bar{b}c) + (ab\bar{c} + abc) \\ = \bar{a}\bar{b}(\bar{c} + c) + a\bar{b}(\bar{c} + c) + ab(\bar{c} + c) \\ = \bar{a}\bar{b} + a\bar{b} + ab = (\bar{a}\bar{b} + a\bar{b}) + (a\bar{b} + ab) \\ = (\bar{a} + a)\bar{b} + a(\bar{b} + b) = \bar{b} + a. \end{aligned} \qquad 3.3$$

Z funkcji 3.2 wynika przy wykorzystaniu wzoru 2.10

$$y = (a + \bar{b} + c)(a + \bar{b} + \bar{c}) = a + \bar{b} + (c \cdot \bar{c}) = a + \bar{b}. \qquad 3.4$$

Zapisu funkcji $f(a, b, c)$ podanej w przykładzie można dokonać również w tzw. tablicy Karnaugha [czyt. *Karno*]. Ma ona postać:

a \ bc	00	01	11	10
0	1	1	0	0
1	1	1	1	1

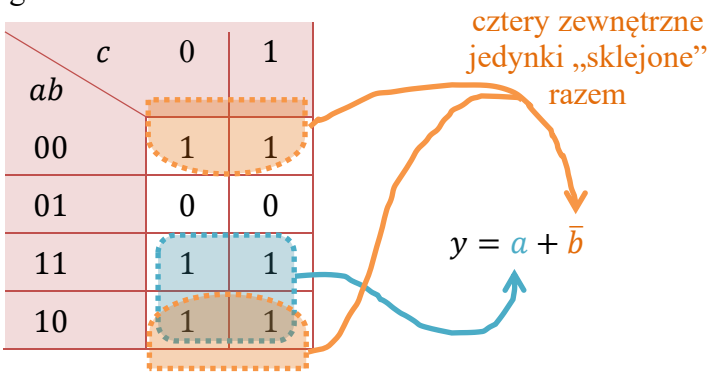
lub

ab \ c	0	1
00	1	1
01	0	0
11	1	1
10	1	1

Uwaga: Należy zwrócić uwagę, że wartości par zmiennych nie są przyjęte jako kolejne liczby dwójkowe (00, 01, 10, 11), ale wg kodu 00, 01, 11, 10 (tzw. *kod Graya*), gdzie sąsiednie kody (również skrajne) różnią się tylko na jednej pozycji. Wynika to ze stosowania reguł sklejania.

W tablicy umieszcza się wartości funkcji $f(a,b,c)$ dla wartości zmiennych a,b,c zaznaczonych na zewnątrz tabeli. Przeprowadzając następnie minimalizację zapisu funkcji na podstawie tablicy Karnaugha otrzymujemy:

a) minimalizując wg stanów 1



b) minimalizując wg stanów 0

ab \ c	0	1
00	1	1
01	0	0
11	1	1
10	1	1

$y = a + \bar{b}$

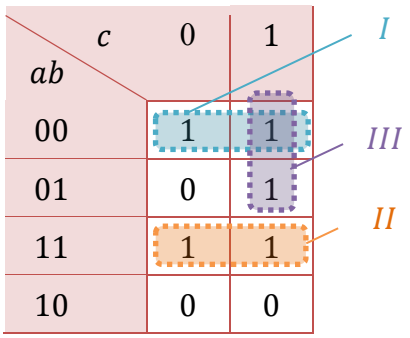
Jak widać zapisy te są identyczne tożsamościowo z obiema postaciami kanonicznymi funkcji, co już wcześniej wykazano we wzorach 3.3 oraz 3.4.

Uwaga: Należy też zwrócić uwagę, że wartości par zmiennych nie są przyjęte jako kolejne liczby dwójkowe (00, 01, 10, 11), ale wg kodu 00, 01, 11, 10 (tzw. kod Graya), gdzie sąsiednie kody (również skrajne) różnią się tylko na jednej pozycji. Wynika to ze stosowania reguł sklejania.

3.2 Przykład 4

Zminimalizować funkcję $f(a,b,c)$ wg podanej tablicy Karnaugha.

Minimalizacja „jedynkowa”



$y = \bar{a}\bar{b} + ab + \bar{a}c$
$I \quad II \quad III$

Każdej parze sąsiadujących jedynek obwiedzionych owalną linią na tablicy Karnaugha odpowiada jeden składnik w zapisie alternatywnym funkcji będący iloczynem dwóch zmiennych, które dla danej pary mają stałe wartości logiczne zaznaczone na zewnątrz tablicy, i tak:

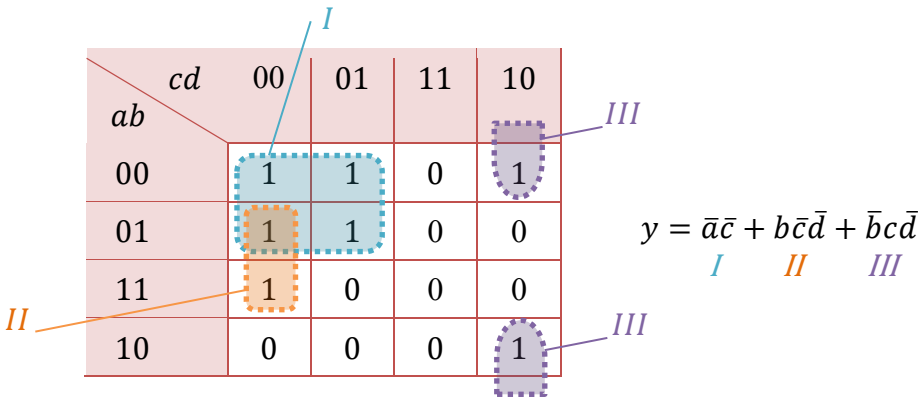
– dla I pary stałe wartości posiadają zmienne $a = 0$ i $b = 0$; I parze odpowiada zatem składnik $\bar{a}\bar{b}$; jest to po prostu graficzna metoda sklejania 2 koniunkcji (wzór 2.10) $\bar{a}\bar{b}\bar{c} + \bar{a}\bar{b}c = \bar{a}\bar{b}(\bar{c} + c) = \bar{a}\bar{b}$;
– dla II pary stałe wartości mają zmienne $a = 1$ i $b = 1$ – odpowiada jej zatem składnik ab, ponieważ $ab\bar{c} + abc = ab(\bar{c} + c) = ab$;
– dla III pary stałe wartości mają zmienne $a = 0$ i $c = 1$ – odpowiada jej zatem składnik $\bar{a}c$, ponieważ $\bar{a}\bar{b}c + \bar{a}bc = \bar{a}c(\bar{b} + b) = \bar{a}c$.

Alternatywa, czyli suma logiczna wymienionych 3 składników jest minimalnym zapisem funkcji przedstawionej tablicą Karnaugha w powyższym przykładzie.

3.3 Przykład 5

Zminimalizować funkcję czterech zmiennych $f(a, b, c, d)$ wg tablicy Karnaugha.

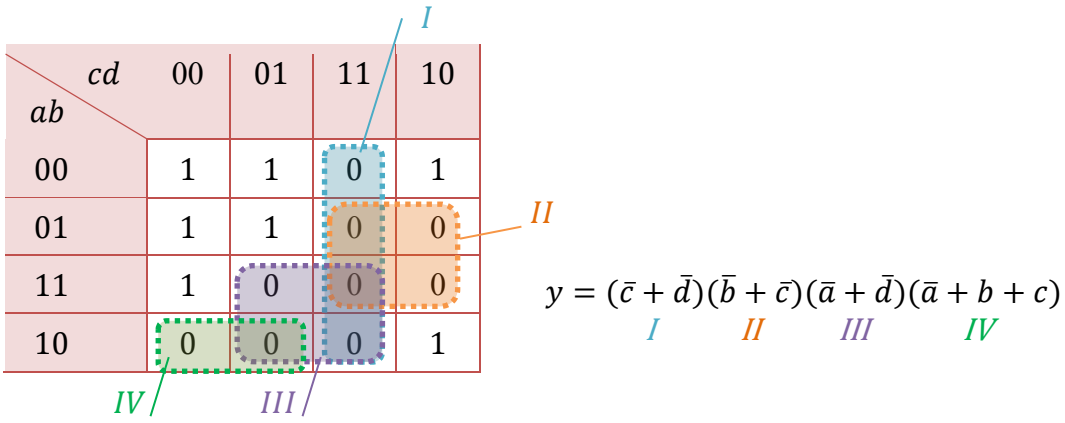
Minimalizacja „jedynkowa”



Objaśnienie: każdej grupie sąsiadujących jedynek obwiedzionych linią przerywaną na tablicy Karnaugha odpowiada jeden składnik w zapisie alternatywnym będący iloczynem zmiennych, które dla danej grupy mają stałe wartości logiczne, i tak:

– dla grupy I (cztery jedynki) stałe wartości posiadają zmienne $a = 0$ i $c = 0$, zatem odpowiada jej zapis $\bar{a}\bar{c}$;
– dla grupy II (dwie jedynki) stałe wartości posiadają zmienne $b = 1$, $c = 0$ i $d = 0$, zatem zapis odpowiadający tej grupie będzie $b\bar{c}\bar{d}$;
– dla grupy III (dwie jedynki zewnętrzne) stałe wartości posiadają zmienne $b = 0$, $c = 1$ i $d = 0$, zatem odpowiada jej zapis $\bar{b}c\bar{d}$.

Minimalizacja „zerowa”



Objaśnienie: każdej grupie sąsiadujących zer obwiedzionych linią przerywaną na tablicy Karnaugha odpowiada jeden czynnik w zapisie koniunkcyjnym funkcji, będący sumą logiczną zmiennych, które mają stałe wartości logiczne dla danej grupy oznaczonej na zewnątrz tablicy; tak więc w powyższym przykładzie:

- dla grupy I zmienne posiadające stałe wartości to: $c = 1$ i $d = 1$, odpowiadający jej zapis jest więc $(\bar{c} + \bar{d})$; wynika to z reguły sklejania, wg wzoru 2.10

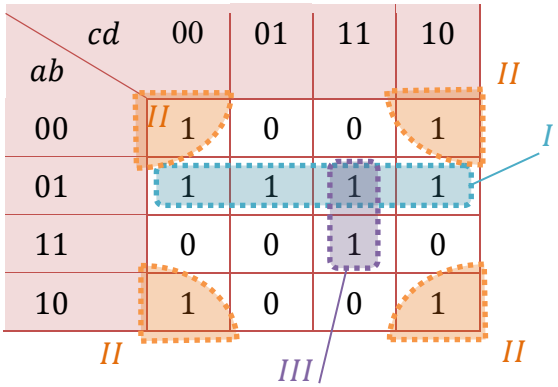
$$(a + b + \bar{c} + \bar{d})(a + \bar{b} + \bar{c} + \bar{d})(\bar{a} + \bar{b} + \bar{c} + \bar{d})(\bar{a} + b + \bar{c} + \bar{d}) \overset{\text{wg wzoru 2.8}}{=}$$
$$= [(a + \bar{c} + \bar{d}) + (b + \bar{b})][(\bar{a} + \bar{c} + \bar{d}) + (b + \bar{b})] =$$
$$= (a + \bar{c} + \bar{d})(\bar{a} + \bar{c} + \bar{d}) = (\bar{c} + \bar{d}) + (a \cdot \bar{a}) = (\bar{c} + \bar{d}),$$

- dla II grupy zmiennymi o stałych wartościach są: $b = 1$ i $c = 1$ – zapis więc będzie $(\bar{b} + \bar{c})$,
- dla III grupy zmiennymi o stałych wartościach są: $a = 1$ i $d = 1$ – zapis więc będzie $(\bar{a} + \bar{d})$,
- dla IV grupy zmienne o stałych wartościach to: $a = 1, b = 0$ i $c = 0$ – zapis więc będzie $(\bar{a} + b + c)$.

3.4 Przykład 6

Przedstawić minimalizację zapisu funkcji $f(a, b, c, d)$ dla podanej tablicy Karnaugha.

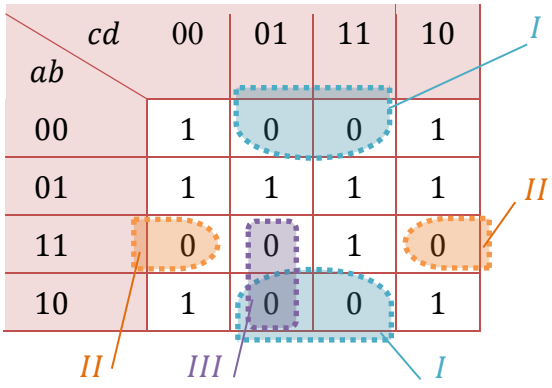
Minimalizacja „jedynkowa”



$$y = \bar{a}b + \bar{b}\bar{d} + bcd$$
$$I \quad II \quad III$$

Na zwrócenie uwagi w tym przykładzie zasługuje grupa II złożona z czterech jedynek umieszczonych na rogach tablicy, dla której zmienne posiadające stałe wartości logiczne to $b = 0$ i $d = 0$.

Minimalizacja „zerowa”

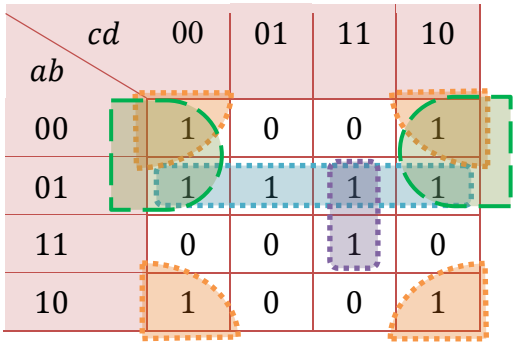


$$y = (b + \bar{d})(\bar{a} + \bar{b} + d)(\bar{a} + c + \bar{d})$$
$$I \quad II \quad III$$

Funkcje uzyskane z minimalizacji „zerowej” i „jedynkowej” są tożsamościowo równe, co można zawsze wykazać stosując reguły rachunku Boole’a.

Na podstawie przytoczonych przykładów można się zorientować, że grupy jedynek lub zer mogą obejmować regularne pola złożone wyłącznie z 2, 4 lub 8 (lub ogólnie z 2^n, gdzie $n \in \mathbb{N}$) kratek tablicy Karnaugha, będące prostokątami lub kwadratami bądź też mogą być rozdzielone na połówki i wówczas te połówki grupy muszą przylegać do zewnętrznych boków tablicy naprzeciwko siebie albo też może być to grupa obejmująca cztery kratki rogowe tablicy Karnaugha (jak w powyższym przykładzie). Pola nie mogą więc obejmować np. 3, 5, 6 czy 7 jedynek lub zer. Może natomiast wystąpić przypadek, że będzie tylko jedna jedynka lub jedno zero. Zawsze należy odnaleźć pola obejmujące możliwie najwięcej kratek, bo wtedy minimalizacja jest najlepsza, a uzyskana funkcja najprostsza.

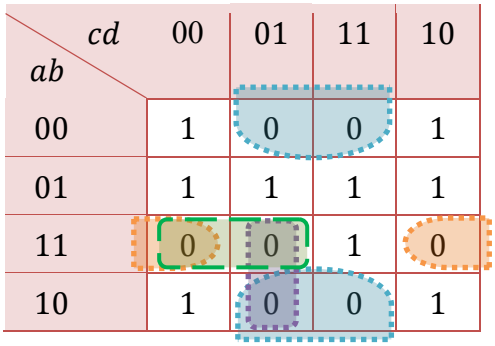
Przy okazji można dostrzec, że pola te często zachodzą na siebie. Fakt ten ma swoje pozytywne znaczenie, bowiem zachodzenie na siebie pól obejmujących grupy jednakowych wartości logicznych jest zjawiskiem korzystnym – przyczynia się do eliminacji niekorzystnych zjawisk, takich jak *wyścig* lub *hazard*. Brak takiego zachodzenia na siebie pól może być niekiedy sztucznie likwidowany przez tworzenie dodatkowych grup łączących sąsiadujące, lecz nie zachodzące na siebie pola, czyli innymi słowy wzbogacenie minimalnego zapisu funkcji o dodatkowy składnik lub czynnik nie wprowadzający zmian w opisie funkcji, za to likwidujący niejednoznaczność funkcji w momencie zmiany stanu jednej ze zmiennych; np. dla przykład 6 takimi składnikami przy minimalizacji jedynkami byłby składnik $\bar{a}\bar{d}$;



wówczas funkcja (już nie minimalna) miałaby zapis:

$$y = \bar{a}b + \bar{b}\bar{d} + bcd + \bar{a}\bar{d}.$$

Przy minimalizacji zerami rolę takiego „łącznika” pełniłby czynnik $(\bar{a} + \bar{b} + c)$;



zapis funkcji byłby wtedy:

$$y = (b + \bar{d})(\bar{a} + \bar{b} + d)(\bar{a} + c + \bar{d})(\bar{a} + \bar{b} + c).$$

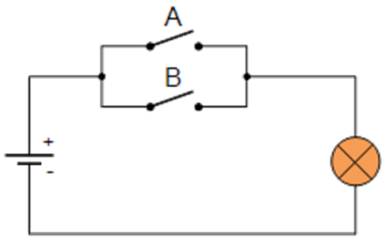
W przypadku opisu funkcji y pięcioma lub więcej zmiennymi minimalizacja jej za pomocą tablicy Karnaugha jest bardziej skomplikowana i nie będzie tutaj omówiona.

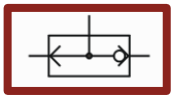
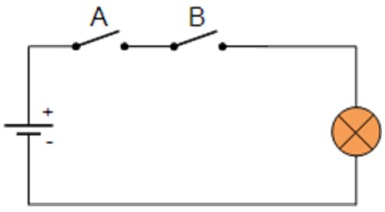
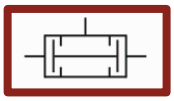
4 Fizyczna realizacja funkcji logicznych

Jak już wspomniano we wstępie układy logiczne mogą być realizowane w praktyce w oparciu o elementy elektroniczne (półprzewodnikowe), elektromechanicznie lub pneumatycznie.

W poniższej tabelce przedstawiono fizyczną realizację funkcji logicznych alternatywy oraz koniunkcji z użyciem elementów elektromechanicznych (zestyki, przełączniki czynne) oraz pneumatycznych (zawory logiczne).

Tabela 2 Fizyczne elektromechaniczne i pneumatyczne realizacje funkcji alternatywy i koniunkcji

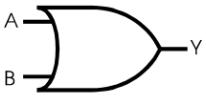
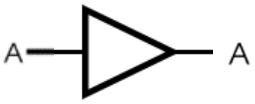
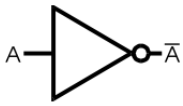
<table>
<tr><th>Elektromechanicznie</th><th></th><th>Pneumatycznie</th></tr>
<tr>
<td> 

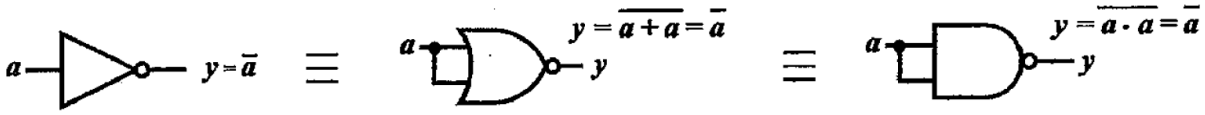
Przełącznik A:
otwarty = „0”, zamknięty = „1”
Przełącznik B:
otwarty = „0”, zamknięty = „1”
Żarówka:
zapalona = „1”, zgaszona = „0”</td>
<td>
<table>
<tr><th>a</th><th>b</th><th>a + b</th></tr>
<tr><td>0</td><td>0</td><td>0</td></tr>
<tr><td>0</td><td>1</td><td>1</td></tr>
<tr><td>1</td><td>0</td><td>1</td></tr>
<tr><td>1</td><td>1</td><td>1</td></tr>
</table>
</td>
<td>Zawór
“LUB”
 
 
</td>
</tr>
<tr>
<td> 
</td>
<td>
<table>
<tr><th>a</th><th>b</th><th>a · b</th></tr>
<tr><td>0</td><td>0</td><td>0</td></tr>
<tr><td>0</td><td>1</td><td>0</td></tr>
<tr><td>1</td><td>0</td><td>0</td></tr>
<tr><td>1</td><td>1</td><td>1</td></tr>
</table>
</td>
<td>Zawór
“I”
 
 
</td>
</tr>
</table>

W technice cyfrowej ogromną rolę odrywają elementy półprzewodnikowe nazywane *bramkami logicznymi*. Są to najprostsze układy cyfrowe realizujące elementarne funkcje logiczne. Służą do budowy układów logicznych o większej złożoności. W tabela 3 pokazano sposób ich oznaczania na schematach oraz realizowane przez nie operacje logiczne.

W tabeli przedstawiono bramki AND, NAND, OR, NOR, XOR oraz NXOR jako elementy 2-wejściowe. W praktyce są również stosowane elementy z większą liczbą wejść. Jak łatwo zauważyć, kółko przy wyjściu graficznego symbolu elementu oznacza negację realizowanej funkcji. Należy też zauważyć, że negację NOT można też zrealizować np. z dwuwejściowych elementów NOR lub NAND, łącząc razem ich wejścia (rys. 4.1).

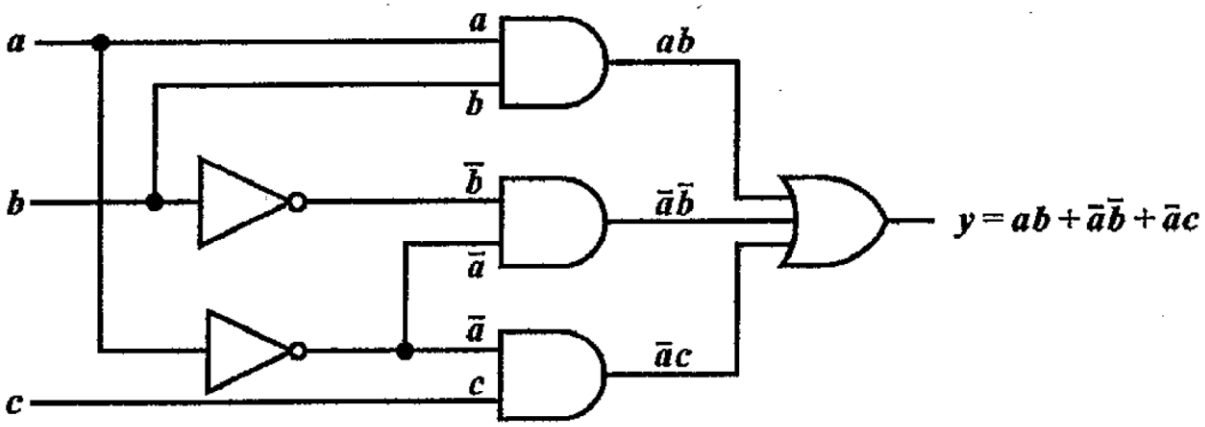
Tabela 3 Podstawowe elementy logiczne (bramki logiczne)

AND	OR	Powtórzenie (wzmacniacz)	XOR
			
$y = a \cdot b$	$y = a + b$	$y = a$	$y = a \oplus b$
NAND	**NOR**	**NOT**	**NXOR**
			
$y = \overline{a \cdot b}$	$y = \overline{a + b}$	$y = \bar{a}$	$y = \overline{a \oplus b}$
Systemy funkcjonalnie pełne			



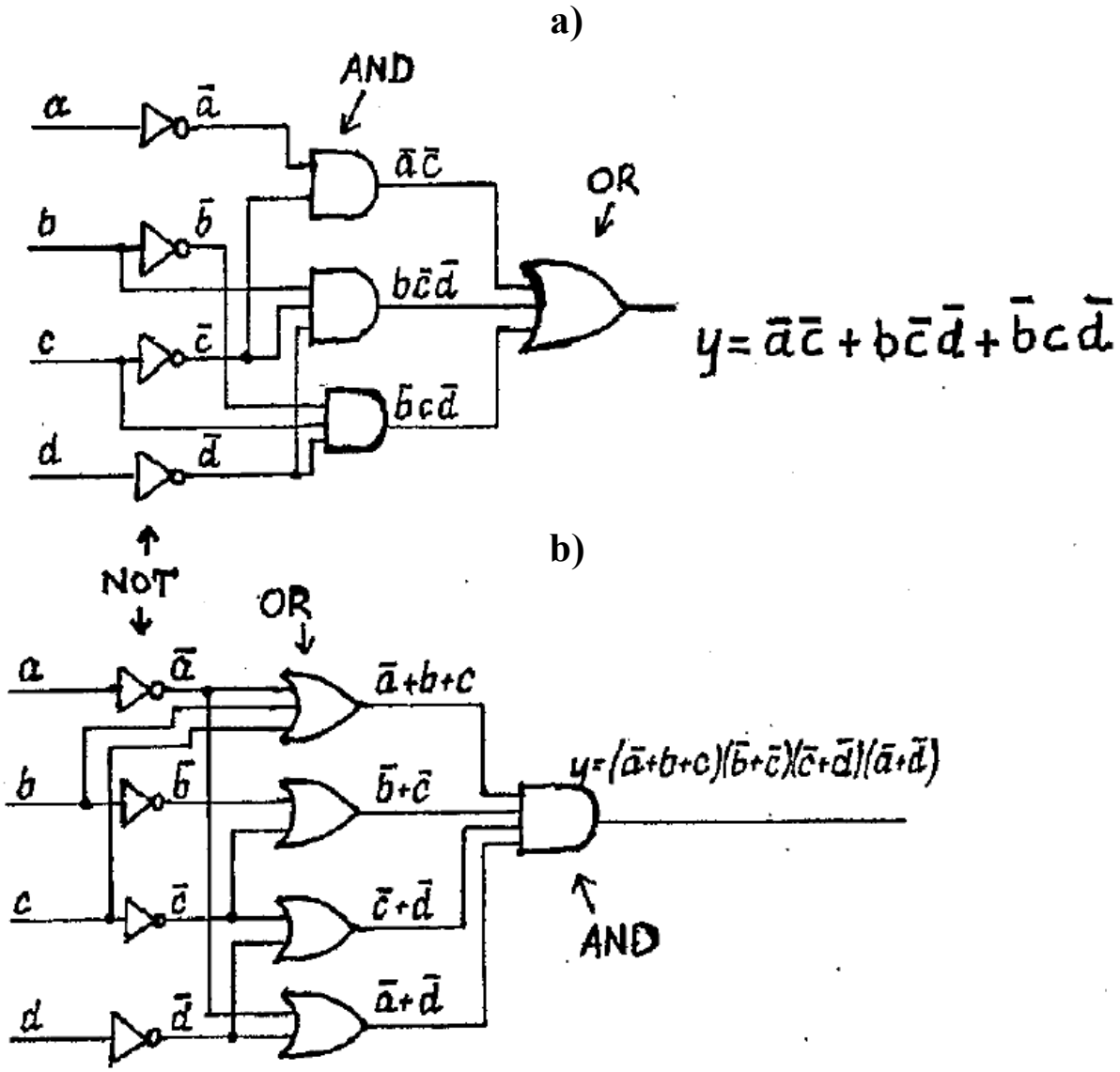
Rys. 4.1 Równoważne realizacje negacji z bramek NOR lub NAND

Schemat realizacji funkcji z przykład 4 (uzyskanej z minimalizacji „jedynkowej") z wykorzystaniem podstawowych elementów logicznych przedstawia poniższy rys. 4.2.



Rys. 4.2 Realizacja funkcji z przykład 4

Schematy realizacji funkcji z przykład 5 przedstawiają rys. 4.3 a) (minimalizacja „jedynkowa") oraz rys. 4.3 b) (minimalizacja „zerowa").



Rys. 4.3 Realizacja funkcji z przykład 5 dla minimalizacji jedynkowej (a) oraz minimalizacji zerowej (b)

W technice są stosowane systemy funkcjonalnie pełne, czyli oparte na jednym bazowym elemencie. Do takich elementów zaliczają się NOR i NAND. Aby zbudować schemat używając wyłącznie elementów, np. NOR należy dokonać przekształcenia zapisu funkcji do postaci zawierającej wyłącznie zaprzeczenie sum logicznych; zatem np. funkcja z przykład 6 o postaci

$$y = \bar{a}b + \bar{b}\bar{d} + bcd + \bar{a}\bar{d}$$

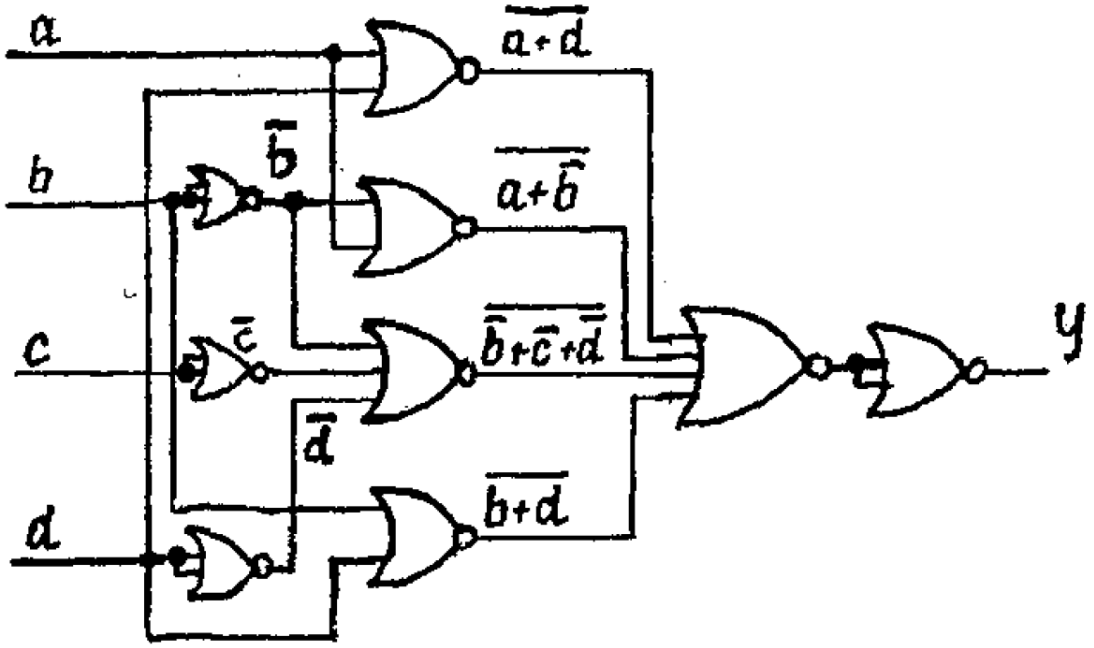
Musi zostać przekształcona przez stosowanie podwójnej negacji i praw de Morgana; zatem

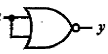
$$y = \overline{\overline{\bar{a}b}} + \overline{\overline{\bar{b}\bar{d}}} + \overline{\overline{bcd}} + \overline{\overline{\bar{a}\bar{d}}} =$$

$$= \overline{\left(\bar{\bar{a}} + \bar{b}\right)} + \overline{\left(\bar{\bar{b}} + \bar{\bar{d}}\right)} + \overline{\left(\bar{b} + \bar{c} + \bar{d}\right)} + \overline{\left(\bar{\bar{a}} + \bar{\bar{d}}\right)} =$$

$$= \overline{\overline{\left(a + \bar{b}\right)} + \overline{(b + d)} + \overline{\left(\bar{b} + \bar{c} + \bar{d}\right)} + \overline{(a + d)}}$$

zapisowi temu odpowiada schemat z rys. 4.4.



Rys. 4.4 Realizacja funkcji z przykład 6 (minimalizacja „jedynkowa") w oparciu o bramki NOR (negacje zbudowano z 2- wejściowych bramek NOR )

Zaś tożsamościowo równą funkcję z minimalizacji „zerowej" o postaci

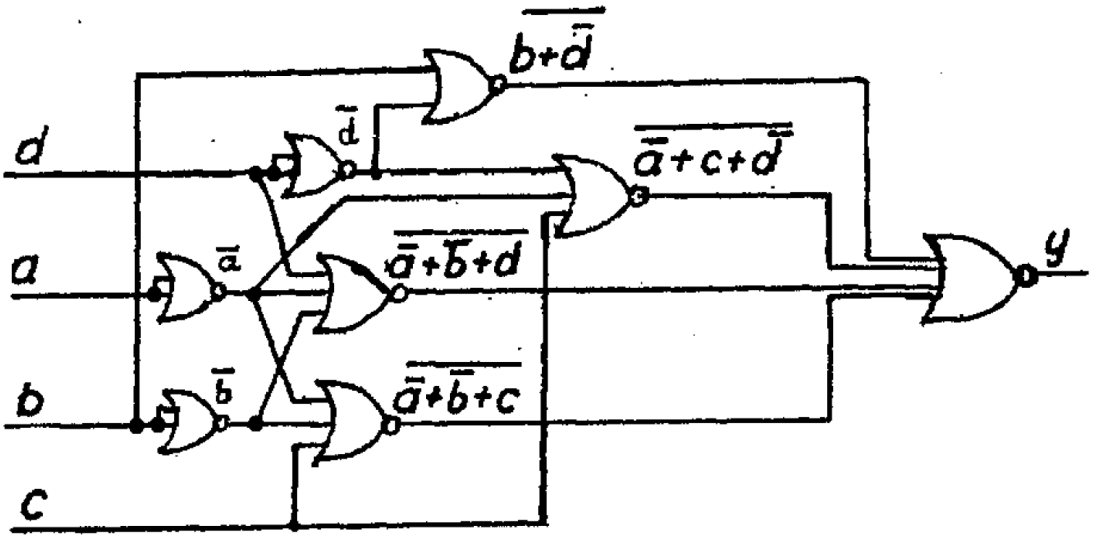
$$y = (b + \bar{d})(\bar{a} + \bar{b} + d)(\bar{a} + c + \bar{d})(\bar{a} + \bar{b} + c)$$

należy przekształcić do zapisu

$$y = \overline{\overline{(b + \bar{d})(\bar{a} + \bar{b} + d)(\bar{a} + c + \bar{d})(\bar{a} + \bar{b} + c)}}$$

$$= \overline{\overline{(b + \bar{d})} + \overline{(\bar{a} + \bar{b} + d)} + \overline{(\bar{a} + c + \bar{d})} + \overline{(\bar{a} + \bar{b} + c)}}$$

wówczas odpowiada jej schemat z rys. 4.5.



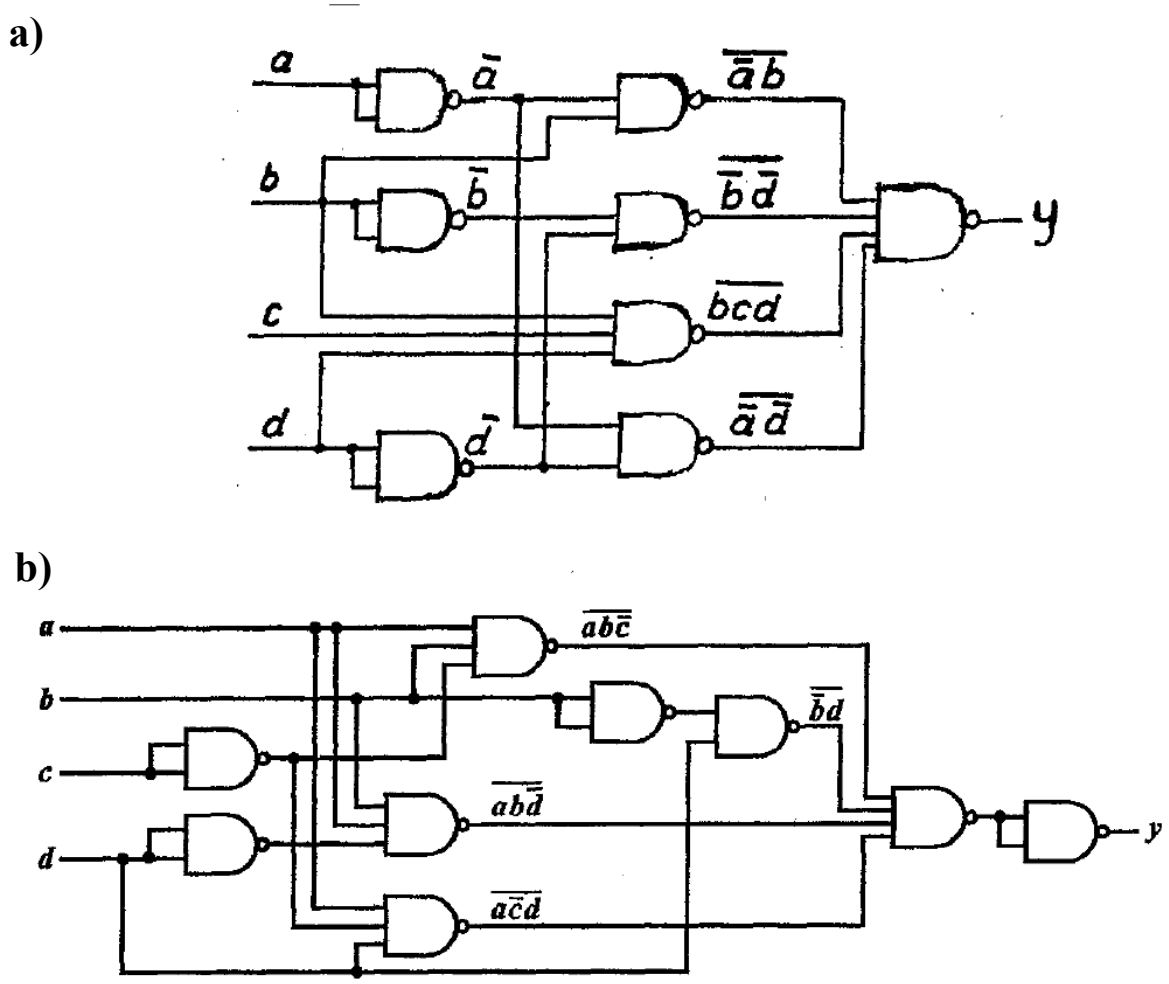
Rys. 4.5 Realizacja funkcji z przykład 6 (minimalizacja „zerowa") zbudowana z NOR

Budując schemat dla realizacji funkcji z elementów NAND należy zapis funkcji przekształcić do postaci zawierającej wyłącznie zaprzeczenia iloczynów logicznych; zatem funkcje z przykład 6 przekształcamy następująco:

$$y = \bar{a}b + \bar{b}\bar{d} + bcd + \bar{a}\bar{d} = \overline{\overline{\bar{a}b + \bar{b}\bar{d} + bcd + \bar{a}\bar{d}}} = \overline{(\overline{\bar{a}b})(\overline{\bar{b}\bar{d}})(\overline{bcd})(\overline{\bar{a}\bar{d}})} \ ,$$

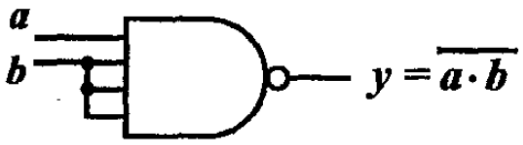
$$\begin{aligned} y = (b + \bar{d})(\bar{a} + \bar{b} + d)(\bar{a} + c + \bar{d})(\bar{a} + \bar{b} + c) = \\ = \left(\overline{\overline{b + \bar{d}}}\right)\left(\overline{\overline{\bar{a} + \bar{b} + d}}\right)\left(\overline{\overline{\bar{a} + c + \bar{d}}}\right)\left(\overline{\overline{\bar{a} + \bar{b} + c}}\right) = \\ = \left(\overline{\bar{b}\bar{d}}\right)\left(\overline{\bar{a}\bar{b}\bar{d}}\right)\left(\overline{\bar{a}c\bar{d}}\right)\left(\overline{\bar{a}\bar{b}c}\right) = \overline{\overline{(\bar{b}\bar{d})(\overline{ab\bar{d}})(\overline{a\bar{c}d})(\overline{ab\bar{c}})}} \ ; \end{aligned}$$

ich schematy odpowiadające końcowym zapisom funkcji przedstawiono na rys. 4.6.



Rys. 4.6 Realizacje funkcji z przykład 6 w oparciu o bramki NAND dla minimalizacji jedynkowej (a) oraz zerowej (b)

Podczas praktycznej realizacji funkcji z dostępnych elementów logicznych może się okazać, że wejścia niektórych bramek pozostaną niepodłączone (np. mamy do dyspozycji bramkę 4-wejściową, a potrzebne są tylko 2 wejścia). W takiej sytuacji najlepiej zewrzeć niewykorzystane wejścia razem z dowolnym sygnałem podanym na jedno z wykorzystanych wejść, jak pokazano na poniższym rysunku.



W praktyce układy logiczne realizuje się współcześnie w technologii TTL z wymaganym napięciem zasilania 5V (np. układy cyfrowe serii 74LS… o małym poborze mocy i opóźnieniu zbocza 9ns) lub w technologii CMOS (np. popularna seria 40… o dopuszczalnym napięciu zasilania 3-15V i opóźnieniu ok. 40ns).

Funkcje logiczne można realizować również na drodze elektromechanicznej, wykorzystując elementy przekaźnikowe stykowe. Pojedynczy element przekaźnikowy może realizować tylko funkcję powtórzenia lub negacji w zależności jak jest zbudowany. Przekaźnik z zestykiem czynnym realizuje funkcję powtórzenia, tzn. zestyk jest zwarty, gdy na cewkę przekaźnika jest podane napięcie. Przekaźnik z zestykiem biernym realizuje funkcję negacji, tzn. zestyk jest zwarty wówczas, gdy na cewce przekaźnika nie ma napięcia. W chwili, gdy na cewce przekaźnika jest napięcie zestyk jest rozwarty.

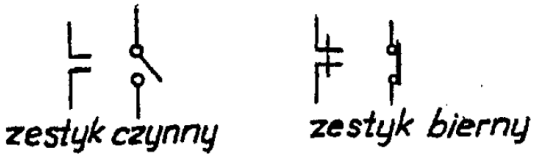
Stosując elementy przekaźnikowe można zbudować dowolną funkcję logiczną.

Na rys. 4.7 przedstawiono symbole stosowane na schemacie układów przekaźnikowych zestyków czynnych i biernych. Na rys. 4.8 przedstawiono schemat realizacji 3 podstawowych funkcji logicznych przy wykorzystaniu elementów przekaźnikowych:

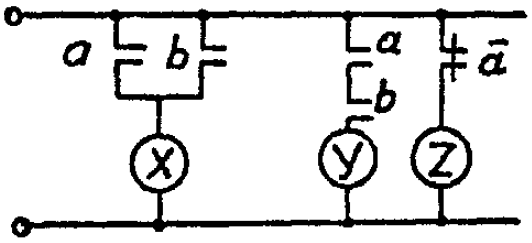
$$x = a + b,$$

$$y = ab,$$

$$z = \bar{a}.$$



Rys. 4.7



Rys. 4.8

W praktyce elementy przekaźnikowe posiadają zarówno zestyki czynne, jak i zestyki bierne sterowane jedną cewką.

5 Układy kombinacyjne

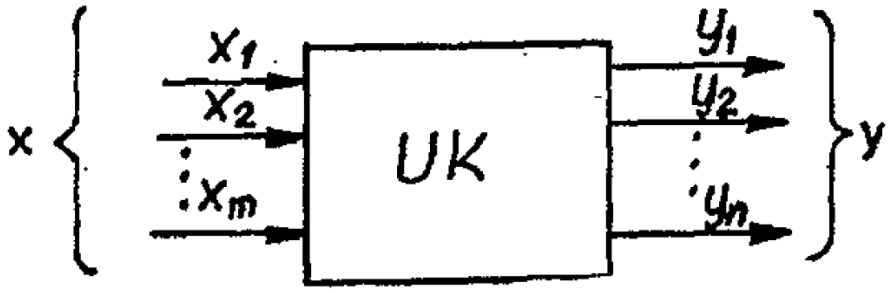
Układem kombinacyjnym nazywamy taki układ przełączający, w którym sygnały wyjściowe, inaczej stan jego wyjść Y, zależą wyłącznie od kombinacji sygnałów wejściowych, tzn. od stanu jego wejść X. Działanie takiego układu nie zależy od czasu. Na aktualny stan wyjścia $Y(t)$ nie ma wpływu jego stan poprzedni $Y(t-T)$. Można więc zapisać

$$Y = f(X),$$

gdzie: Y – stan wszystkich wyjść układu $(y_1, y_2, \dots, y_n)$,

X – stan wszystkich wejść układu $(x_1, x_2, \dots, x_m)$.

Ogólny schemat układu kombinacyjnego UK przedstawia poniższy rys. 5.1.



Rys. 5.1

Projektowanie układu kombinacyjnego składa się z następujących etapów:

1) na podstawie słownego sformułowania zadań wykonywanych przez układ określa się liczbę jego wejść i wyjść,
2) zestawia się tablicę zależności stanów wyjść od stanów wejść,
3) na podstawie tablicy zależności znajduje się wyrażenie strukturalne dla każdego wyjścia; ponieważ w układach przełączających sygnały mogą przyjmować wartości 0 lub 1, wyrażenia te będą funkcjami Boole'owskimi,
4) upraszcza się wyrażenie strukturalne jedną z metod minimalizacji funkcji Boole'a,
5) na podstawie uproszczonych wyrażeń strukturalnych rysuje się schemat połączeń układu,
6) sprawdza się prawidłowość działania układu z założeniami.

Sposób projektowania układu rozpatrzymy na poniższym przykładzie.

5.1 Przykład 7

Należy zaprojektować układ sterujący silnikami wentylatorów W_1 i W_2 w hali fabrycznej w zależności od temperatury pomieszczenia. Temperatura jest mierzona przez 3 termometry kontaktowe T_1, T_2, T_3 nastawione na 3 różne temperatury: $T_1 = 25℃$, $T_2 = 30℃$, $T_3 = 35℃$. Moc wentylatorów: $W_1 = 5$ kW, $W_1 = 10$ kW.

Założenia:

- gdy temperatura pomieszczenia t jest mniejsza od 25℃ wentylatory nie pracują,
- gdy $25℃ \leq t < 30℃$ pracuje wentylator W_1,
- gdy $30℃ \leq t < 35℃$ pracuje wentylator W_2,
- gdy $t \geq 35℃$ pracują oba wentylatory,
- nieprawidłowe działanie termometrów powoduje uruchomienie alarmu A.

Rozwiązanie:

1) Liczba wejść układu – 3 (T_1, T_2, T_3) oraz liczba wejść układu – 3 (W_1, W_2, A). Przyjęto ponadto następujące oznaczenia: sygnał na wyjściu termometru, gdy temperatura osiągnie lub przekroczy wartość nastawioną przyjmuje stan 1; stan 1 oznacza również, że wentylator lub alarm pracują.
2) Tablicę zależności po przyjęciu takich oznaczeń przedstawiono poniżej.

Ilość wierszy tablicy zależy od liczby wejść układu, gdyż musi ona przedstawiać działanie układu dla wszystkich kombinacji sygnałów wejściowych. W tym przykładzie ilość wierszy wynosi 2^3. Stany przedstawione w wierszach 1, 2, 3, 5 nie mogą wystąpić przy prawidłowym działaniu termometrów, więc dla wentylatorów są to stany obojętne; tzn. można je wykorzystać do minimalizacji jako zera lub jako jedynki, tzn. nie narzucamy układowi żadnych wymagań dotyczących pracy wentylatorów w przypadku uszkodzeń termometrów. Zakładamy, że w tym przypadku sygnalizacja A spowoduje ingerencję człowieka (ręczne sterowanie wentylatorami).

Lp.	Wejścia układu			Wyjścia układu			
	T_1	T_2	T_3	W_1	W_2	A	
0	0	0	0	0	0	0	$\leftarrow t < 25℃$
1	0	0	1	–	–	1	
2	0	1	0	–	–	1	
3	0	1	1	–	–	1	
4	1	0	0	1	0	0	$\leftarrow 25℃ \leq t < 30℃$
5	1	0	1	–	–	1	
6	1	1	0	0	1	0	$\leftarrow 30℃ \leq t < 35℃$
7	1	1	1	1	1	0	$\leftarrow t \geq 35℃$

3) Tablice Karnaugha dla każdego wyjścia otrzymane na podstawie tablicy zależności przedstawiono poniżej.

T_1T_2 \ T_3	0	1
00	0	–
01	–	–
11	0	1
10	1	–

W_1

T_1T_2 \ T_3	0	1
00	0	–
01	–	–
11	1	1
10	0	–

W_2

T_1T_2 \ T_3	0	1
00	0	1
01	1	1
11	0	0
10	0	1

A

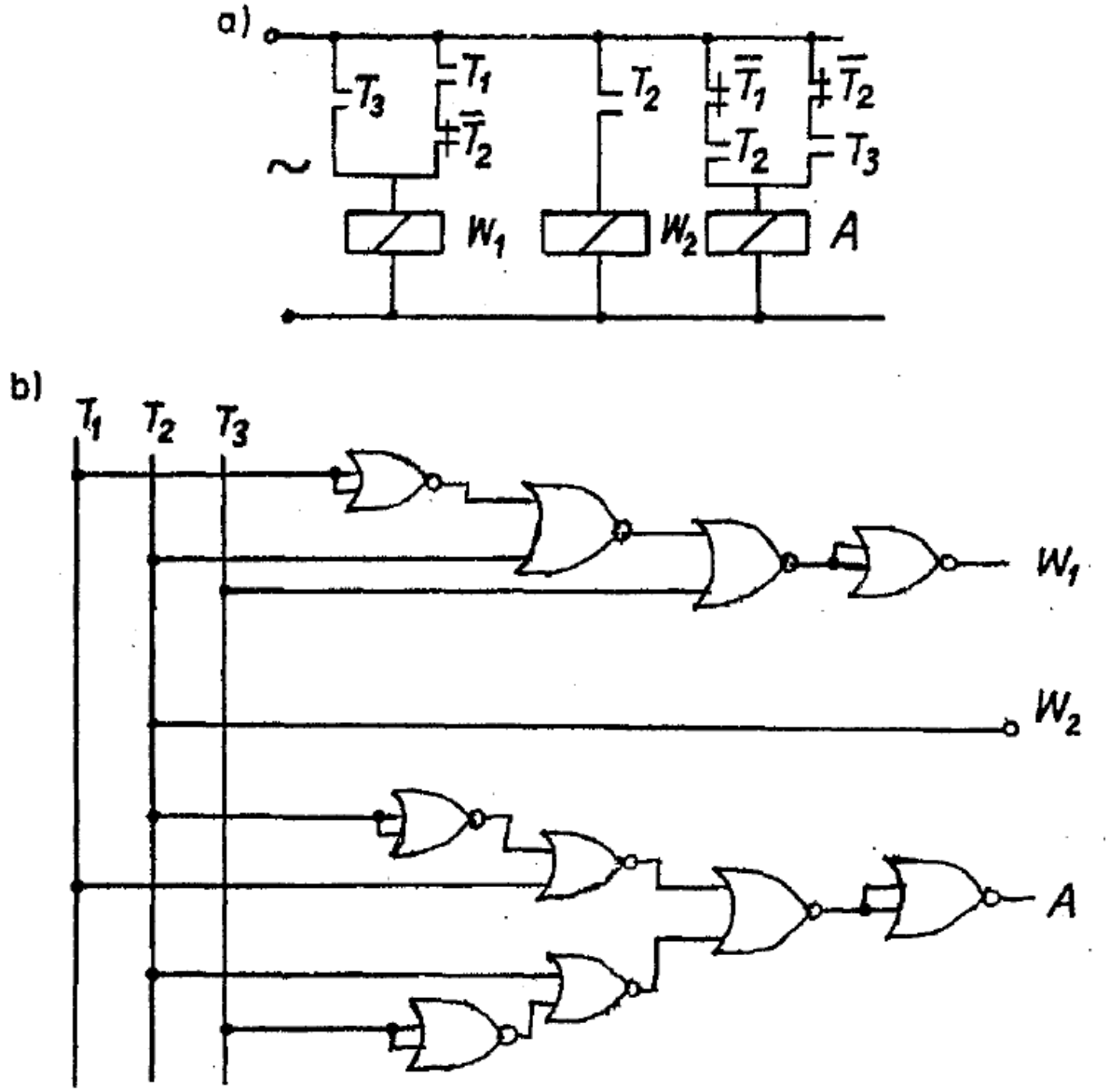
Po minimalizacji wyrażeń otrzymujemy następujące zależności:

$$W_1 = T_3 + T_1\overline{T_2},$$

$$W_2 = T_2,$$

$$A = \overline{T_1}T_2 + \overline{T_2}T_3.$$

4) Poniższy rysunek przedstawia schemat połączeń rozpatrywanego układu zbudowanego z elementów stykowych (a) oraz z elementów NOR (b).

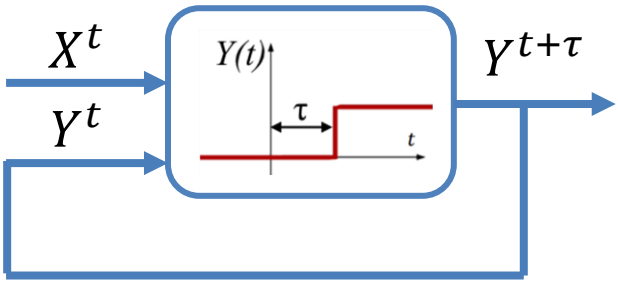


5) Analiza obu schematów potwierdza spełnienie wszystkich założeń do przykładu.

6 Układy logiczne sekwencyjne

W poprzednim rozdziale były omawiane układy logiczne, w których stan wyjść układu $\boldsymbol{Y} = (y_1, y_2, ...)$ zależał jedynie od aktualnego stanu (chwilowej **kombinacji**) wejść układu $\boldsymbol{X} = (x_1, x_2, ...)$. Takie układy nazywane są układami **kombinacyjnymi.** W technice występują sytuacje, w których znajomość kombinacji wejść może być niewystarczająca do określenia wyjścia. Za przykład niech posłuży nam automat do napojów i przekąsek[1]. Jeżeli użytkownik ma ochotę na przekąskę umieszczoną pod numerem „12", to wciśnie na klawiaturze przyciski z numerami „1" i „2". Jednak mając ochotę na napój spod numeru „21" użytkownik wciska te same dwa przyciski tylko w odwrotnej kolejności – najpierw „2", a później „1". Oznacza to, że do wyboru odpowiedniego produktu (określenia wyjścia układu logicznego) potrzebna jest znajomość nie tylko kombinacji przycisków (wejść układu) ale też **sekwencja** ich zmiany. Układy logiczne realizujące takie zadania nazywane są **sekwencyjnymi.** Uwzględnienie sekwencji (historii zdarzeń) zachodzącej na wejściach układu jest możliwe dzięki wprowadzeniu do układu **pamięci.** W tej pamięci nie przechowuje się wszystkich zmian zachodzących na wejściach układu, ale jedynie tzw. **wewnętrzny stan** układu logicznego.

Aby zrozumieć podstawowy mechanizm działania pamięci, w analizie działania układów logicznym musimy uwzględnić **czas.** Fizycznie wykonany układ logiczny (nie zależnie od technologii wykonania) będzie „wykonywać obliczenia" przez jakiś skończony czas, tzn. od momentu pojawienia się nowej kombinacji wejść do pojawienia się nowego stanu wyjść upłynie czas τ. Dlatego, każdy element układu logicznego możemy traktować jako człon wprowadzający opóźnienie. Podstawowym sposobem na wprowadzenie pamięci do układu jest wykorzystanie sprzężenia zwrotnego sygnału wyjściowego, jak pokazano to na schemacie na rys. 6.1. Jeżeli w takim układzie w chwili t, pojawi się nowa kombinacja sygnałów wejściowych $\boldsymbol{X}^t$, to nowy stan wyjść $\boldsymbol{Y}^{t+\tau}$ pojawi się dopiero po upływie czasu opóźnienia (czasu obliczeń) w chwili czasowej $t + \tau$. Do tego momentu układ „pamięta" stan wyjść z chwili początkowej $\boldsymbol{Y}^t$ (i zostaje on wykorzystany w obliczeniach).



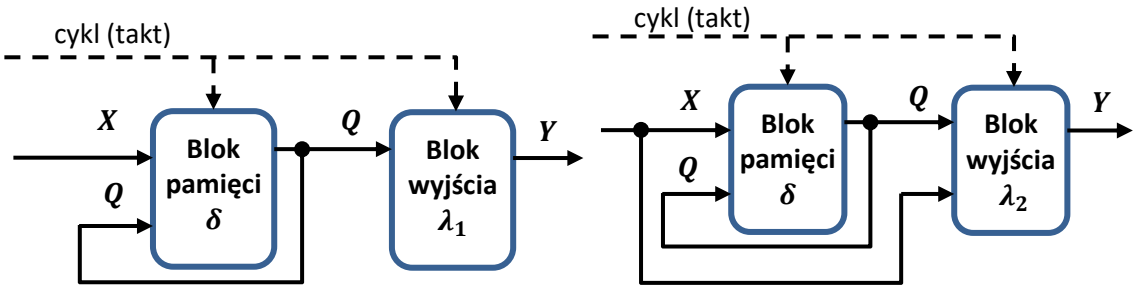
Rys. 6.1 Elementarny układ sekwencyjny

Przy projektowaniu układów logicznych interesuje nas głównie ich stan ustalony, tzn. po upływnie opóźnienia τ. Dlatego układy sekwencyjne analizuje się w **czasie dyskretnym**, czyli w chwilach czasowych po kolejnych opóźnieniach nazywanych **taktami**. Czas opóźnienia upływający pomiędzy kolejnymi taktami może mieć różne wartości i wynikać ze skumulowanego opóźnienia jego elementów składowych (układy asynchroniczne) lub być regulowany przez zewnętrzy układ zegarowy (układy synchroniczne). Wartość tego czasu nie jest istotna w naszych rozważaniach, dlatego stan układu w danym takcie będziemy oznaczać indeksem t $(\boldsymbol{X}^t, \boldsymbol{Y}^t)$, w takcie następnym indeksem $t + 1$ $(\boldsymbol{X}^{t+1}, \boldsymbol{Y}^{t+1})$, a w poprzednim $t - 1$ $(\boldsymbol{X}^{t-1}, \boldsymbol{Y}^{t-1})$.

[1] Współczesne automaty do przekąsek mają układy sterowania programowane na uniwersalnych sterownikach lub innych układach mikroprocesorowych, prawdopodobnie z wykorzystaniem programowej instrukcji wielokrotnego wyboru „switch". Przykład ten proszę traktować jako uproszczony, o dużym stopniu ogólności.

6.1 Podstawowe struktury układów sekwencyjnych - automaty Moore'a i Mealy'ego

W najprostszej wersji, funkcje logiczne układów sekwencyjnych można syntezować (wyprowadzać) stosując metody wcześniej opisane dla układów kombinacyjnych, przyjmując zmienne wyjściowe (lub określające wewnętrzny stan układu) $\boldsymbol{Y}^{t+1}$ jako wyjście układu, a $\boldsymbol{Y}^{t}$ jako wejście. W ten sposób tworząc niezbędne sprzężenie zwrotne. Jednak sprzężenie zwrotne sprawia, że stany przejściowe układu (które były pomijane przy analizie układów kombinacyjnych) mogą wywoływać nieporządne efekty, powodujące nieprawidłowe działanie układu. Dlatego synteza układów sekwencyjnych wymaga większej uważności. Podstawowe struktury wg. jakich projektuje się układy sekwencyjne to tzw. automaty Moore'a i Mealy'ego. Ich schematy blokowe są przedstawione na rys. 6.2.



a) b)

Rys. 6.2 Podstawowe struktury układów sekwencyjnych: a) automat Moore'a i b) automat Mealy'ego.

Na powyższych schematach wektor wejść układu jest opisany jako $\boldsymbol{X} = (x_1, x_2, ...,)$, wektor wyjść $\boldsymbol{Y} = (y_1, y_2, ...)$, a wektor reprezentujący wewnętrzy stan układu (stan pamięci) to $\boldsymbol{Q} = (q_1, q_2, ...)$. To właśnie zmienne $(q_1, q_2, ...)$ są przekazywane w sprzężeniu zwrotnym realizując pamięć układu. Wracając do przykładu automatu z przekąskami, w stanie pamięci może być przechowywana informacja o tym ile i które przyciski zostały już wybrane, a stan wyjść będzie uruchamiał silnik na odpowiedniej półce.

W logice działania obu automatów możemy wyróżnić dwa etapy: określenie nowego stanu automatu oraz określenie na jego podstawie nowego stanu wyjść. Każdy z tych etapów jest realizowany przez układ kombinacyjny, odpowiednio w bloku pamięci lub bloku wyjścia. Nowy stan pamięci układu $\boldsymbol{Q}^{t+1}$ jest obliczany w bloku pamięci, a obliczenia te przebiegają zgodnie z funkcją przejść δ, na podstawie aktualnego stanu wejść $\boldsymbol{X}^{t}$ oraz aktualnego stanu pamięci $\boldsymbol{Q}^{t}$:

$$\boldsymbol{Q}^{t+1} = \delta(\boldsymbol{Q}^{t}, \boldsymbol{X}^{t}). \qquad 6.1$$

Następnie w bloku wyjścia, obliczany jest stan zmiennych wyjściowych $\boldsymbol{Y}^{t}$ zgodnie z funkcją wyjść λ. Tutaj pojawia się różnica pomiędzy opisywanymi automatami. W automacie Moore'a, argumentem funkcji wyjść jest tylko stan wewnętrzny układu:

$$\boldsymbol{Y}^{t} = \lambda_1(\boldsymbol{Q}^{t}), \qquad 6.2$$

natomiast w automacie o strukturze Mealy'ego stan wyjść jest określany zarówno na podstawie aktualnego stanu pamięci $\boldsymbol{Q}^{t}$ jak i stanu wejść $\boldsymbol{X}^{t}$:

$$\boldsymbol{Y}^{t} = \lambda_2(\boldsymbol{Q}^{t}, \boldsymbol{X}^{t}). \qquad 6.3$$

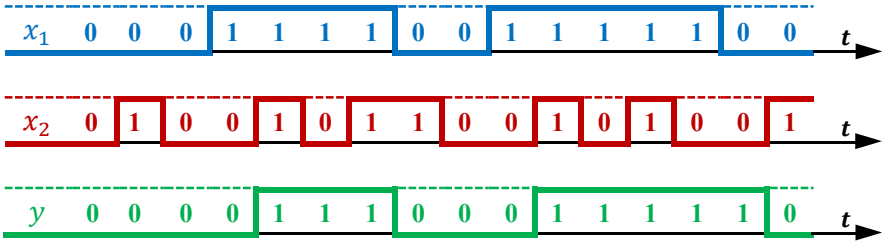
Na obu powyższych schematach (Rys. 6.2) występuje jeszcze sygnał cyklu (taktu). Nie jest on elementem koniecznym. Jest to sygnał zegarowy (ciąg impulsów prostokątnych) wyznaczający chwile czasowe, w których do układu wpuszczane są nowe stany wejść oraz w których wystawiane są nowe stany wyjść. Układy posiadające taki sygnał nazywane są **synchronicznymi**,

natomiast układy bez tego sygnału to układy **asynchroniczne.** Podstawową różnicą pomiędzy tymi układami jest to, że w układach asynchronicznych sygnał o stanie wysokim (1) lub niskim (0) uważany jest za jeden sygnał, bez względu na czas trwania. Natomiast w układach synchronicznych sygnał trwający n taktów sygnału zegarowego, jest uważany za n kolejnych sygnałów.

6.2 Sposoby opisywania układów sekwencyjnych

Najpowszechniejszym sposobem opisu układów sekwencyjnych jest **opis słowny**, przyporządkowujący stany zmiennych wyjściowych do sekwencji stanów wejściowych. Opis ten, w zależności od skomplikowania układu może być zwięzły lub bardzo rozbudowany. Trudno jednak na podstawie samego opisu słownego napisać funkcję logiczną, dlatego opis słowny zazwyczaj wykorzystuje się do opracowania kolejnych opisów układu, bardziej użytecznych przy syntezie funkcji logicznych.

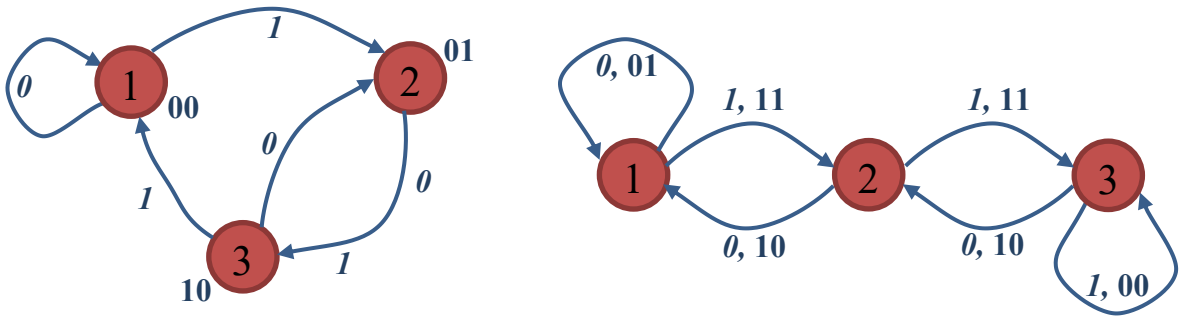
Kolejnym sposobem przedstawienia sposobu działania układu są **ciągi zero-jedynkowe** i **wykresy czasowe.** Przedstawiają one typową sekwencję działania układu, przypisując kolejnym stanom wejściowym, odpowiednie stany wyjść. Przykład takiego opisu przedstawia rys. 6.3. Jak widać, na takich wykresach oś czasu zazwyczaj nie jest wyskalowana, a sygnały przyjmują tylko wartości 1 lub 0, z pominięciem wszelkich stanów przejściowych.



Rys. 6.3 Wykres czasowy i ciągi zero-jedynkowe opisujące układ sekwencyjny.

Opis słowny i wykresy czasowe opisują zależności pomiędzy wejściami $\boldsymbol{X}$ i wyjściami układu $\boldsymbol{Y}$. Jednak aby zaprojektować układ w strukturze Moore'a czy Mealy'ego potrzebujemy znać informacje o stanie pamięci $\boldsymbol{Q}$ oraz o funkcjach przejść δ i wyjść λ. Stosowane są dwie podstawowe metody opisów układów uwzględniające te informacje – graficzna i tabelaryczna.

Metoda graficzna polega na wykreśleniu **grafu układu**. Wierzchołki takiego grafu reprezentują stany pamięci $\boldsymbol{Q}$, natomiast gałęzie grafu reprezentują stan wejść $\boldsymbol{X}$, powodujący przejście pomiędzy dwoma stanami. Gałąź grafu jest skierowana od jednego stanu, do stanu w którym układ ma się znaleźć po pojawieniu się stanu wejściu $\boldsymbol{X}$, co opisuje funkcję przejść δ. W zależności od struktury automatu inaczej na grafie zaznaczane są stany wyjść $\boldsymbol{Y}$. W automacie Moore'a stan wyjść zależy tylko od stanu pamięci układu, dlatego wpisuje się je przy wierzchołkach grafu. W układzie automaty Mealy'ego stany wyjść zależą zarówno od stanu pamięci jak i stanu wejść, np. $\boldsymbol{Y_1 = \lambda_2(Q_2, X_3)}$**.** Taki zapis oznacza, że jeżeli układ jest w stanie $\boldsymbol{Q_2}$, a na wejściach jest stan $\boldsymbol{X_3}$, to wyjścia powinny być w stanie $\boldsymbol{Y_1}$. Na grafie $\boldsymbol{Y_1}$ wpisuje się obok gałęzi $\boldsymbol{X_3}$, wychodzącej z wierzchołka $\boldsymbol{Q_2}$. Przykładowe grafy dla automatu Moore'a i Mealy'ego są przedstawione na rys. 6.4. Stany wyjść określają tam dwie zmienne binarne, tzn. $\boldsymbol{Y} = (y_1, y_2)$, a stan wejść jedna - $\boldsymbol{X} = (x)$. W obu przyjęto trzy stany pamięci: $\boldsymbol{Q_1, Q_2, Q_3}$, które na grafach zaznaczono po prostu jako $1, 2, 3$.



a) b)

Rys. 6.4 Grafy dla układu Moore'a (a) i Mealy'ego (b).

Tabelarycznym odpowiednikiem grafów układu są **tablice przejść i wyjść.** Są one mniej obrazowe, ale za to zdecydowanie wygodniejsze przy przekształceniach i wyprowadzaniu zminimalizowanych funkcji logicznych. Zgodnie z nazwą, tablica przejść opisuje funkcję przejść δ. Zgodnie z równaniem (4.1), każdej parze $(\boldsymbol{Q}^t, \boldsymbol{X}^t)$, przypisuje nowy stan pamięci $\boldsymbol{Q}^{t+1}$. Taką tablice dla automatu Moore'a zadanego grafem z Rys. 6.4a przedstawia tablica z rys. 6.5a, natomiast rys. 6.5b przedstawia tablicę przejść dla automatu Mealy'ego z rys. 6.4b.

Q^t \ X^t	0	1	Y^t
1	1	2	**00**
2	3	3	**01**
3	2	1	**10**
	Q^{t+1}		

a)

Q^t \ X^t	0	1
1	1	2
2	1	3
3	2	3
	Q^{t+1}	

b)

Q^t \ X^t	0	1
1	01	11
2	10	11
3	10	00
	Y^t	

c)

Rys. 6.5 Tablica przejść i wyjść układu Moore'a oraz tablice przejść (b) i wyjść (c) układu Mealy'ego.

Funkcja wyjść dla automatu Moore może być opisana jedną kolumną (Rys. 6.5a), a dla układu Mealy'ego będzie to oddzielna tablica (Rys. 6.5c).

Tablice przejść i wyjść są formą opisu układów sekwencyjnych, która jest najbardziej przydatna przy dalszych przekształceniach, ale ich tworzenie nie jest oczywiste ponieważ początkowo (w opisie słownym) nie mamy informacji o wewnętrznych stanach układu. Dlatego zbudowanie grafów układu oraz tablic przejść i wyjść stanowi początkowy etap syntezy sekwencyjnego układu logicznego.

Określenie stanów wewnętrznych jest etapem wymagającym największego udziału myślenia kreatywnego. Co prawda istnieją sformalizowane metody określania stanów wewnętrznych (tzw. algebra zdarzeń), ale są one skomplikowane a ich poznanie i stosowanie pracochłonne. W układach o stosunkowo małym stopniu skomplikowania poprawne rezultaty można zazwyczaj uzyskać posługując się metodami intuicyjnymi.

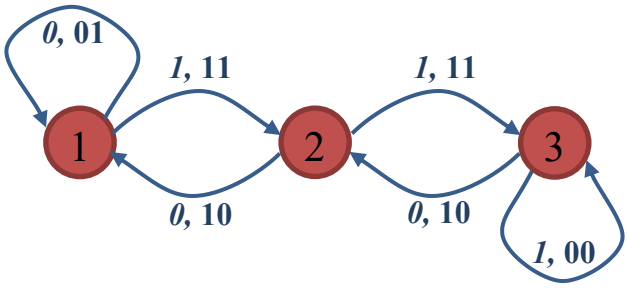
Najprostszym przypadkiem są układy o z góry znanej pamięci. Przykładem takiego układu może być urządzenie, którym nie można sterować bez uprzedniego podania hasał. W takim układzie będą występować jedynie dwa stany wewnętrze, do zakodowania których wystarczy nam tylko jedna zmienna binarna np. „przed wpisaniem hasła" - stan $\boldsymbol{Q}_1$ = (0) i „po wpisaniu

hasła" stan $\boldsymbol{Q}_2$ = (1). Innym przykładem może być układ uruchamiający silnik, który ma wykonać określoną liczbę obrotów. Wtedy w stanach wewnętrznych będzie przechowywana informacja o liczbie wykonanych obrotów.

Innym sposobem może być analiza wykresów czasowych (lub ciągów zero-jedynkowych) i poszukiwanie stanów w których mamy ten sam układ wejść a inny stan wyjść. Każdą taką sytuację możemy rozróżnić poprzez wprowadzanie kolejnych stanów wewnętrznych. Nie należy obawiać się wprowadzenia zbyt dużej liczby stanów wewnętrznych, ponieważ analizując tablicę przejść łatwo jest zidentyfikować stany zgodne i następnie je zredukować.

6.3 Synteza układów asynchronicznych

W układach asynchronicznych każda zmiana stanu wejściowego powoduje bezpośrednio zmiany stanu pamięci oraz stanu wyjść. Rozważmy układ dany grafem z Rys. 6.4a (lub tablicą z Rys. 6.5a). Zgodnie z grafem, jeżeli układ znajduje się w stanie 1, a na wejściu jest 0, to układ pozostanie w stanie 1. Jednak jeżeli na wejściu pojawi się 1, to układ przejdzie do stanu 2, następnie do stanu 3, do stanu 1 i ponownie do 2. Jak długo na wejściu będzie 1, tak długo układ będzie się ciągle przełączał pomiędzy stanami wewnętrznymi. Oznacza to, że są to **stany niestabilne**. Poprawne działanie sekwencyjnego układu asynchronicznego wymaga obecności **stanów stabilnych,** tzn. takich które nie zmieniają się przy stałym sygnale wejściowym $\boldsymbol{Q}^{t+1} = \boldsymbol{Q}^{t}$. Taka sytuacja dla powyższego grafu zachodzi w stanie 1 przy wejściu 0. Dla układu danego grafem z rys. 6.4b, nie znajdziemy takiego ciągu niekończących się zmian stanu układu. Tam dla każdego stanu wejść można znaleźć stabilny stan układu. Stany stabilne zaznaczamy w tablicy przejść otaczając je okręgiem, tak jak to przedstawiono na rys. 6.6b. Żeby układ asynchroniczny mógł działać poprawnie, dla każdej kombinacji stanów wejściowych, musi mieć przynajmniej jeden stan stabilny.



a)

Q^t \ X^t	0	1
1	(1)	2
2	1	3
3	2	(3)
	Q^{t+1}	

b)

Rys. 6.6 Grafy dla układu Mealy'ego (a) i jego tablica przejść z zaznaczonymi stanami stabilnymi (b).

6.3.1 Hazard statyczny

Jak było wspomniane wcześniej, każdy element układu realizującego funkcję logiczną wprowadza opóźnienie do układu. Jeżeli układ będzie zrealizowany w technice cyfrowej, to sygnały elektryczne reprezentujące stany logiczne na poszczególnych wejściach bramki pojawią się w różnych momentach. Oznacza to, że wynik na wyjściu bramki będzie „nieprawidłowy" do chwili, w której wszystkie sygnały wejściowe będą aktualne. Ten „nieprawidłowy" stan będzie trwał tylko przez czas równy różnicy w czasie przesyłania sygnałów wejściowych. W układzie kombinacyjnym to zjawisko zazwyczaj nie stanowi problemów. Jednak w układzie sekwencyjnym, ten „nieprawidłowy" stan może zostać przekazany przez sprzężenie zwrotne na wcześniejszy etap obliczeń i w efekcie zostać podtrzymany lub zafałszować inne obliczenia.

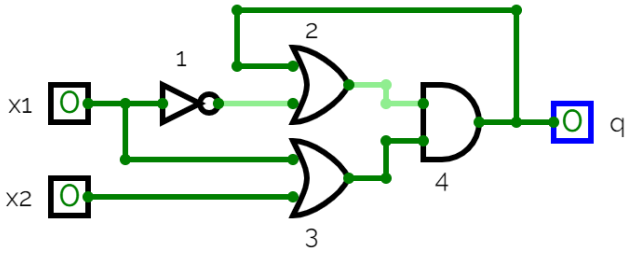
Rozważmy tablicę przejść z rys. 6.7a. Stan wejść opisują dwie zmienne (x_1, x_2), natomiast stan pamięci jest dany przez jedną zmienną q. Funkcję przejść zapiszemy wykorzystując metodę minimalizacji zerowej:

$$q^{t+1} = (x_1 + x_2) \cdot (q + \bar{x}_1) \qquad 6.4$$

Schemat układu logicznego zbudowanego na podstawie równania (4.4) prezentuje Rys. 6.7b.

$(q)^t$ \ $(x_1, x_2)^t$	00	01	11	10
0	0	1	0	0
1	0	1	1	1
	q^{t+1}			

a)



b)

Rys. 6.7 Tablica przejść (a) i odpowiadający jej układ logiczny (b) z ryzykiem hazardu statycznego.

Jak można zaobserwować sygnał x_1 wpływa na stan q dwiema drogami o różnej liczbie elementów. Jedna droga zawiera dwie bramki: OR (3) i AND (4), natomiast druga zawiera trzy bramki: NOT (1), OR (2) i AND (4). Rozpatrzmy sytuację, w której $(x_1, x_2) = (0,0)$, $q = 0$. Następnie x_1 zmienia wartość na 1. Zgodnie z tablicą przejść nowy stan $q^{t+1} = 0$. Nowy stan zmiennej x_1 najpierw pojawia się na bramce OR (3), co sprawia że na jej wyjściu pojawi się stan wysoki – 1, który przekazany na końcową bramkę AND (4), spowoduje zmianę jej wyjścia na stan wysoki $q = 1$. Ta wartość przez sprzężenie zwrotne pojawia się na bramce OR (2), na której z opóźnieniem spowodowanym przez bramkę NOT (1), pojawia się dopiero nowa wartość $\overline{x_1} = 0$. Na wyjściu bramki (3) pojawia się w takim razie stan wysoki, co powoduje podtrzymanie nieprawidłowej wartości na wyjściu $q = 1$.

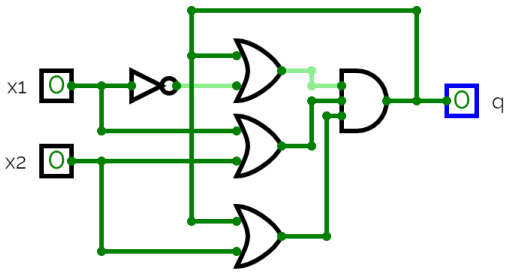
Opisane zjawisko polegające na pojawieniu się niewłaściwego stanu q na skutek różnicy w czasie przesyłania sygnału po dwóch różnych drogach nazywa się **hazardem (ryzykiem) statycznym**. Wiąże się on z występowanie w funkcji logicznej tego samego argumentu raz z negacją i raz bez negacji. Stan taki można poznać w tablicy Karnaugha po dwóch grupach kratek stykających się ze sobą na linii zmiany wartości danej zmiennej. W omawianym przypadku problematyczną zmienną jest x_1, linią zmiany wartości jest brzeg tablicy. Hazard wynika z tego, że elementy układu realizujące jeden składnik funkcji jeszcze nie zaczęły działać, a elementy odpowiedzialne za kolejny składnik funkcji już przestały. Wynika z tego, ze funkcja logiczna zagrożona hazardem statycznym potrzebuje jeszcze jednego składnika, który zabezpieczy jej działanie w tej niepewnej sytuacji. Taki składnik dopiszemy przez zakreślenie dodatkowej grup kratek w tablicy Karnaugha, która połączą dwie wcześniejsze, tak jak to pokazano na rys. 6.8. Funkcja logiczna wolna od hazardu ma następującą postać:

$$q^{t+1} = (x_1 + x_2) \cdot (q + \bar{x}_1) \cdot (q + x_2) \qquad 6.5$$

Omówiony powyżej przykład dotyczył syntezy funkcji logicznej w postaci kanonicznej alternatywnej (z wykorzystaniem minimalizacji zerowej), ale te same reguły dotyczą wykrywania i eliminacji hazardu statycznego dla funkcji logicznych w postaci kanonicznej koniunkcyjnej (minimalizacja jedynkowa).

$(q)^t$ \ $(x_1, x_2)^t$	00	01	11	10
0	0	1	0	0
1	0	1	1	1
	q^{t+1}			

a)



b)

Rys. 6.8 Tablica przejść (a) i odpowiadający jej układ logiczny (b) z wolny od hazardu statycznego.

6.3.2 Kodowanie stanów wewnętrznych i wyścig krytyczny.

Zmienne wejściowe i wyjściowe u układach logicznych są opisane zmiennymi dwuwartościowymi (binarnymi). W układach technicznych zmiennymi wejściowymi mogą być stany przycisków (wciśnięty lub nie) czy stany wszelkich czujników krańcowych wykrywających kontakt lub obecność przedmiotu (jest lub nie ma). Zmienne wyjściowe to będą z kolei zazwyczaj sygnały sterujące układem wykonawczym np.: silnik włączony lub nie, zawór zamknięty lub otwarty, lampa zapalona lub zgaszona, itd. Kwestia wewnętrznych zmiennych stanu układu (pamięci) nie jest już tak oczywista, ponieważ oznaczenia stanów mają charakter umowny. Teoria związana z układami logicznymi była rozwijana razem z rozwojem techniki cyfrowej. Dlatego klasycznie przyjmuje się, że wewnętrzne stany układu koduje się z wykorzystaniem zmiennych binarnych. Przykładowo, do zakodowania czterech stanów pamięci, wystarczą dwie zmienne binarne, które zapewniają cztery możliwe kombinacje (00, 01, 11, 10). Taki sposób kodowania powoduje, że do zaprojektowania poprawnego przełączania pomiędzy stanami, trzeba rozważyć również zależności pomiędzy poszczególnymi zmiennymi. A wybór sposobu zakodowania poszczególnych stanów może mieć znaczący wpływ na złożoność znalezionych funkcji logicznych.

Okazuje się, że opóźnienie wprowadzane przez poszczególne elementy wykonujące funkcje logiczne, ma też wpływ na przełączanie stanów układu. Bezpiecznym wydaje się być założenie, że żadne dwa elementy układu logicznego nie wykonują obliczeń dokładnie w tym samym czasie. Taka obserwacja prowadzi do wniosku, że bardzo mało prawdopodobne jest uzyskanie w układzie fizycznym przełączenia dwóch zmiennych w dokładnie tej samej chwili. Przykładowo, oznacza to że przejście ze stanu (00) do (11) zawsze będzie odbywać się ze stanem pośrednim: (00) → (10) → (11) lub (00) → (01) → (11).

Q^t \ X^t	0	1
00	11	00
01	01	10
11	11	10
10	11	10
	Q^{t+1}	

Rys. 6.9 Tablica przejść z wyścigiem krytycznym.

Biorąc pod uwagę powyższe, przenalizowano tablicę przejść z rys. 6.9. Jeżeli układ jest w stanie $\boldsymbol{Q} = (00)$, a wejścia wynoszą $\boldsymbol{X} = (0)$, to układ powinien przejść do stanu, $\boldsymbol{Q} = (11)$. Jak

już zauważono wcześniej, takie przejście musi odbyć się w dwóch krokach. Poprawne przełączenie jest możliwe w sekwencji (00) → (10) →(11), ponieważ stan $\boldsymbol{Q} = (10)$ dalej kieruje układ do stabilnego stanu $\boldsymbol{Q} = (11)$. Jednak przejście ze stanem pośrednim (01) nie jest możliwe, ponieważ stan $\boldsymbol{Q} = (01)$ jest stanem stabilnym i układ już w nim pozostanie. Takie zjawisko uzyskania błędnego stanu stabilnego na skutek różnicy czasu działania elementów nazywa się **wyścigiem krytycznym.**

Jak można zaobserwować, dla powyższej tablicy przy $\boldsymbol{X} = (1)$ zjawisko wyścigu nie występuje. Jeżeli w projektowanym układzie asynchronicznym występuje wyścig krytyczny, należny zmienić kodowanie poszczególnych stanów lub dodać stan pośredni, aby to zjawisko wyeliminować.

6.3.3 Stany zgodne

Jak napisano wcześniej, określenie stanów wewnętrznych układu może być kłopotliwe. A pierwsza opracowana tablica przejść (tablica pierwotna) może zawierać więcej stanów niż jest to potrzebne do poprawnego działania układu sekwencyjnego. Ogólnie można przyjąć, że im więcej stanów wewnętrznych, tym bardziej skomplikowany układ logiczny. Dlatego korzystnie jest zredukować nadmiarowe stany. Dwa (lub więcej) stany wewnętrze możemy zastąpić jednym, jeżeli oba stany uznamy za **zgodne**. Zgodność najłatwiej jest stwierdzić analizując tablicę przejść.

Dla automatu o strukturze Moore'a za zgodne możemy uznać stany, które spełniają wszystkie poniższe warunki:

- dla tych samych wejść $(\boldsymbol{X_i})$ przechodzą do tych samych (lub zgodnych) stanów (tzn. opisują je jednakowe wiersze w tablicy przejść i wyjść),
- odpowiada im ten sam stan wyjściowy $(\boldsymbol{Y})$,
- jeśli są stabilne przy tym samym $\boldsymbol{X}$, to odpowiadające im stany $\boldsymbol{Y}$ są jednakowe.

Jeżeli w tablicy przejść są niewypełnione pola, (tzn. że stan automatu jest tam nieustalony) to możemy je uzupełnić tak, żeby uzyskać stany zgodne. Spójrzmy na przykład z rys. 6.10. Analizując tablicę pierwotną, możemy zaobserwować, że stany 2 i 4 są zgodne, ponieważ dla każdego stanu wejść (X_1, X_2X_3, X_4), przechodzą do tych samych stanów $\boldsymbol{Q}^{t+1}$ oraz mają te same stany wyjść $\boldsymbol{Y} = (0)$. W zredukowanej tablicy przejść połączeniu stanów 2 i 4 opowiada nowy stan 2.

Q^t \ X^t	X_1	X_2	X_3	X_4	Y^t
1	(1)	3	2	(1)	**1**
2	1	(2)	4	1	**0**
3	(3)	-	4	1	**1**
4	1	2	(4)	1	**0**
	$\boldsymbol{Q}^{t+1}$				

a)

Q^t \ X^t	X_1	X_2	X_3	X_4	Y^t
(1,3)→1	(1)	(1)	2	(1)	**1**
(2,4)→2	1	(2)	(2)	1	**0**
	$\boldsymbol{Q}^{t+1}$				

b)

Rys. 6.10 Redukcja tablicy przejść automatu Moore'a: a) tablica pierwotna, b) tablica zredukowana.

Zastąpić jednym nowym stanem możemy również stany 1 i 3, jednak ten przypadek nie jest taki oczywisty. Stany te są zgodne ponieważ:

- dla $\boldsymbol{X_4}$ przechodzą do tego samego stanu 1,

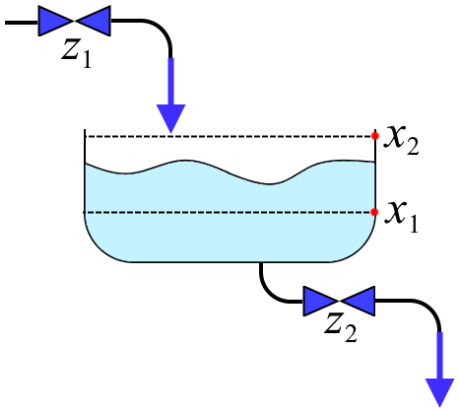
– dla $\boldsymbol{X_3}$ przechodzą od stanów zgodnych 2 i 4,
– dla $\boldsymbol{X_2}$przechodzą do tego samego stanu, jeżeli wykorzystamy stan nieokreślony (-) i wpiszemy w jego miejsce stan 3,
– dla $\boldsymbol{X_1}$ oba stany są stabilne i odpowiada im ten sam stan wyjść $\boldsymbol{Y} = (1)$.

W zredukowanej tablicy przejść połączone stany 1 i 3 tworzą nowy stan 1.

Aby dwa stany uznać za zgodne w strukturze automatu Mealy'ego nie jest wymaga zgodność wyjść $\boldsymbol{Y}$. Podczas syntezy układu sekwencyjnego, pierwotna tablica przejść jest zazwyczaj przygotowana dla struktury Moore'a. Znajdując w niej stany zgodne dla automatu Mealy'ego, można zdecydować o zmianie typu automatu. Trudno jest z góry przewidzieć, która opcja okaże się mniej skomplikowana, dlatego trzeba opracować obie i dopiero dokonać wyboru.

6.3.4 Przykład syntezy układu asynchronicznego w strukturze Moore'a

Przykładowa synteza układu asynchronicznego w strukturze Moore'a zostanie przedstawiona dla układu sterowania poziomem cieczy w zbiorniku. Schemat układu przedstawiono na rys. 6.11. Składa się on ze zbiornika, zaworu wlewowego z_1, zaworu spustowego z_2 oraz z dwóch czujników poziomu cieczy x_1 i x_2. Cieczy do zbiornika można dolewać otwierając zawór z_1 oraz upuszczać otwierając zawór z_2.



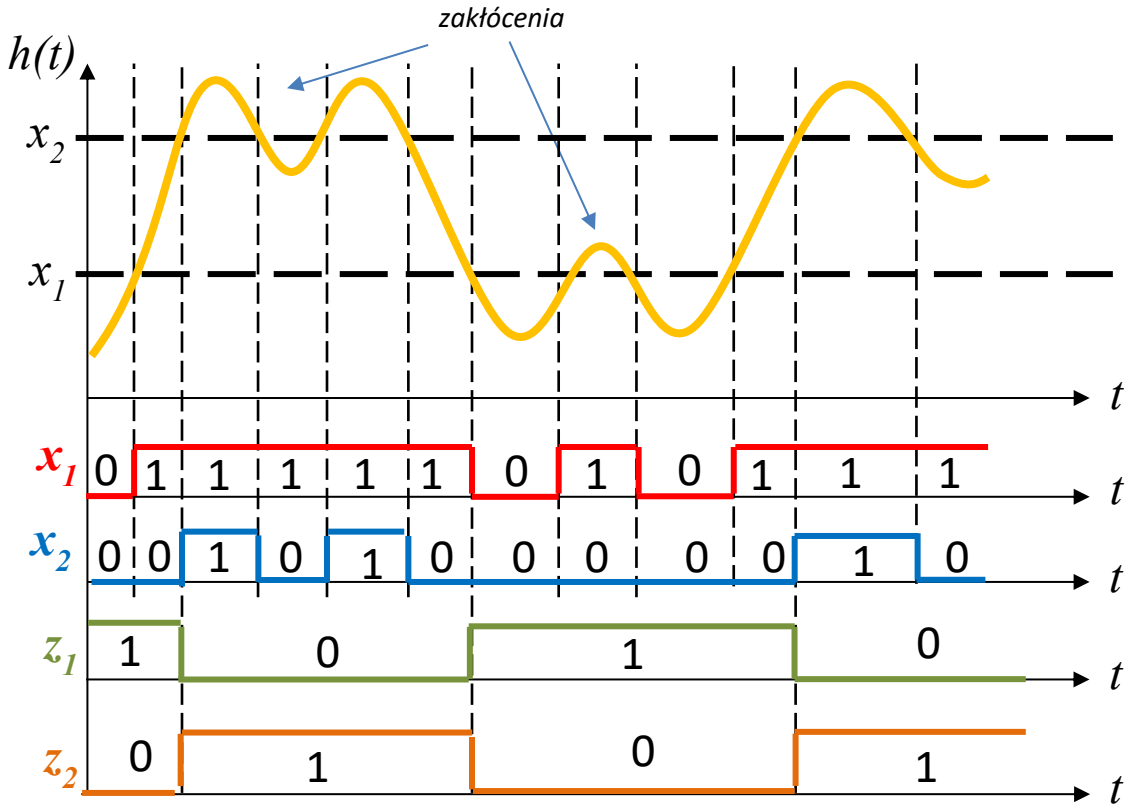
Rys. 6.11 Układ zbiornika z zaworami z_1 i z_2 oraz czujnikami poziomu cieczy x_1 i x_2.

Projekt układu logicznego rozpoczniemy od opisu słownego pożądanego działania układu: *Zadanie polega na utrzymaniu poziomu cieczy w zbiorniku pomiędzy poziomami x_1 i x_2. Działanie układu ma zostać zatrzymane na wypadek awarii czujników.*

W następnej kolejności przygotujemy wykresy czasowe odpowiadające typowemu działaniu (typowej sekwencji) układu. W tym celu najpierw przypiszemy stany logiczne odpowiadające fizycznym stanom układu:

– ciecz poniżej poziomu czujnika → stan logiczny 0,
– ciecz powyżej lub równo z poziomem czujnika → stan logiczny 1,
– zawór otwarty → stan logiczny 1,
– zawór zamknięty → stan logiczny 0.

Wykresy czasowe zostaną przygotowane z pomocą wykresu poziomu cieczy, uwzględniającego zakłócenia – falowanie cieczy, co przedstawia rys. 6.12

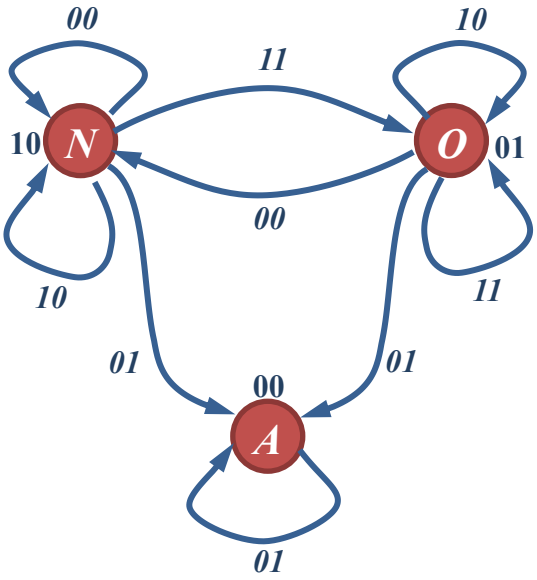


Rys. 6.12 Wykres czasowy typowej sekwencji działania układu.

Analizując wykresy czasowe (lub ciągi zero-jedynkowe) możemy zaobserwować, że realizacja zadania wymaga układu sekwencyjnego, ponieważ przy pośrednim poziome cieczy ($x_1 = 1$, $x_2 = 0$) raz zbiornik jest napełniany a raz opróżniany, tzn. temu samemu stanowi wejść odpowiadają różne stany wyjściowe.

Wektor wartości wejściowych obejmuje stany czujników poziomu cieczy, tj. $\boldsymbol{X} = (x_1, x_2)$. Natomiast wektor wartości wyjściowych będzie opisany przez stany zaworów, tj . $\boldsymbol{Y} = (z_1, z_2)$. Jeżeli chodzi o stany wewnętrzne układu, to intuicyjne można je określić jako stan napełniania zbiornika N i stan opróżniania O. Stanowi napełniania odpowiada stan wyjść $\boldsymbol{Y} = (1, 0)$, a stanowi opróżniania $\boldsymbol{Y} = (0,1)$. Przewidziana miała być również możliwość awaryjnego zatrzymania działania układu w wypadku awarii czujników. Na podstawie odczytu stanu czujników za awarię można uznać sytuację ($x_1 = 0,\ x_2 = 1$), która oznaczałaby poziom cieczy jednocześnie powyżej czujnika x_2 i poniżej czujnika x_1. Zdecydowano, że w takim przypadku należy zamknąć oba zawory. Na taki wypadek wprowadzono stan awarii A, któremu odpowiada stan wyjść $\boldsymbol{Y} = (0, 0)$.

Na podstawie przebiegów czasowych z rys. 6.12, sporządzono graf przejść, który przedstawia rys. 6.13a. Rozpoczęto od zaznaczenia wierzchołków grafu reprezentujących stany N, O i A. Następnie analizując kolejne stany wejściowe dodawano kolejne gałęzie grafu. Inaczej wyglądała analiza stanu awaryjnego. Jego zależności nie wynikają z typowego (poprawnego) działania układu, dla którego sporządzono wykresy czasowe. Tutaj przyjęto założenia, co do tego jak układ ma zadziałać w sytuacji awaryjnej. Przyjęto, że układ wchodzi w stan awarii A, kiedy pojawi się stan wejściowy $\boldsymbol{X} = (0, 1)$.



a)

Q^t \ X^t	00	01	11	10	Y^t
N	(N)	A	O	(N)	**10**
O	N	A	(O)	(O)	**01**
A	-	(A)	-	-	**00**
	Q^{t+1}				

b)

Rys. 6.13 Graf układu (a) oraz odpowiadająca mu tablica przejść i wyjść (b).

Na podstawie grafu układu opracowano pierwotną tablice przejść i wyjść (Rys. 6.13b) z zaznaczonymi stanami stabilnymi. Następnym etapem jest minimalizacja tablicy, przez redukcję stanów zgodnych. W uzyskanej tablicy jednak nie znajdujemy stanów zgodnych dla struktury Moore'a.

Następnie przyjęto sposób kodowania stanów. Do zakodowania trzech stanów potrzebne są dwie zmienne binarne q_1 i q_2. Sposób kodowania przedstawiono w tablicy z rys. 6.14a.

Q	q_1	q_2
N	0	0
O	0	1
A	1	1
-	1	0

a)

Q^t \ X^t	00	01	11	10	Y^t
00	(00)	11	(01)	(00)	**10**
01	00	11	(01)	(01)	**01**
11	-	(11)	-	-	**00**
10	-	-	-	-	**-**
	Q^{t+1}				

b)

Rys. 6.14 Kodowanie wewnętrznych stanów układu (a) oraz odpowiadająca mu tablica przejść i wyjść (b).

Zakodowaną tablice przejść należy przeanalizować pod kątem wyścigów krytycznych. Dla założonego kodowania, problematyczne może być przejście układu ze stanu napełniania $\boldsymbol{Q} = (0,0)$ do stanu awarii $\boldsymbol{Q} = (1,1)$, przy pojawieniu się na wejściu $\boldsymbol{X} = (0,1)$. Poprawne przełączenie jest możliwe w sekwencji (00) → (01) →(11). Aby umożliwić przełączenie w sekwencji (00) → (10) →(11) wykorzystano nieokreślone przejście układu dla $\boldsymbol{Q} = (1,0)$ i $\boldsymbol{X} = (0,1)$. Nie określone przejście zastąpiono przejściem do stabilnego stanu $\boldsymbol{Q} = (1,1)$.

Funkcję przejścia $\boldsymbol{Q}^{t+1} = \delta(\boldsymbol{Q}^t, \boldsymbol{X}^t)$ należy przygotować dla obu zmiennych wewnętrznych q_1 i q_2. W tym celu przygotowano oddzielną tablicą przejść dla każdej zmiennej (patrz Rys. 6.15a i Rys. 6.15b).

Q^t \ X^t	00	01	11	10
00	0	1	0	0
01	0	1	0	0
11	-	1	-	-
10	-	1	-	-
	q_1^{t+1}			

a)

Q^t \ X^t	00	01	11	10
00	0	1	1	0
01	0	1	1	1
11	-	1	-	-
10	-	1	-	-
	q_2^{t+1}			

b)

Q	q_1q_2	z_1	z_2
N	00	1	0
O	01	0	1
A	11	0	0
-	10	-	-

c)

Rys. 6.15 Tablice przejść dla wewnętrznych zmiennych stanu q_1 (a) i q_2 (b) oraz tablica wyjść (c)

Na podstawie tablic zapisano poszczególne funkcje przejść wykorzystując metodę minimalizacji jedynkowej:

$$q_1^{t+1} = \overline{x_1}x_2, \tag{6.6}$$

$$q_2^{t+1} = x_2 + q_2x_1. \tag{6.7}$$

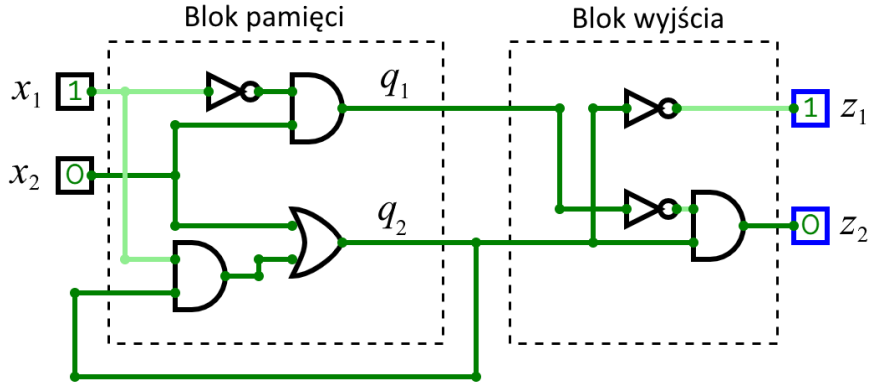
Podczas minimalizacji stany nieokreślone można wykorzystać jako „jedynki" lub zera w zależności od wybranej postaci funkcji logicznej. Należy przy tym pamiętać, że przez taką poszerzoną minimalizację przyjmujemy pośrednio sposób działania układu w stanach, które wcześniej były nieokreślone. Przed podjęciem takiego kroku należy przeanalizować jaki ten zabieg ma wpływ na działanie układu. Analizując wyprowadzone funkcje logiczne, można stwierdzić, że nie występuje w nich zagrożenie hazardem statycznym.

Do wyprowadzenia pozostała jeszcze funkcja wyjść $\boldsymbol{Y}^t = \lambda_1(\boldsymbol{Q}^t)$ na podstawie tablicy z rys. 6.15c:

$$z_1 = \overline{q_2}, \tag{6.8}$$

$$z_2 = \overline{q_1}q_2. \tag{6.9}$$

Układ z bramek logicznych realizujący zaprojektowany układ sekwencyjny (6.6)-(6.9) przedstawia rys. 6.16.



Rys. 6.16 Realizacja zaprojektowanego logicznego układu sekwencyjnego (6.6)-(6.9) z wykorzystaniem bramek logicznych.

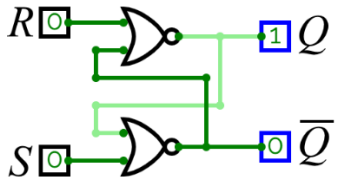
6.4 Synteza układów synchronicznych

Układy synchroniczne charakteryzują się obecnością sygnału wyznaczającego kolejne takty działania układu. Taki sygnał nazywany jest **sygnałem zegarowym, taktującym** lub **synchronizującym** i ma postać ciągu impulsów wyznaczających chwile czasowe w których mogą wystąpić zmiany stanu wejść i pamięci. Takie rozwiązanie powinno wyeliminować problemu hazardu statycznego. Aby układ synchroniczny działał poprawnie, wypadkowe opóźnienie spowodowane działaniem poszczególnych elementów τ powinno być dłuższe od czasu trwania impulsu taktującego, oraz krótsze od czasu impulsu razem z czasem przerwy pomiędzy impulsami. Spełnienie takich warunków może wymagać skracania czasu impulsu taktującego lub wstawiania elementów opóźniających do układu logicznego. W praktyce, tę kwestię rozwiązuje się poprzez wykorzystanie do synchronizacji stanu wejść i pamięci różnych wartości (lub zboczy) sygnału taktującego, np. na elementy wyzwalające zmianę stanu pamięci podaje się odwrócony sygnał zegarowy (np. przez bramkę NOT).

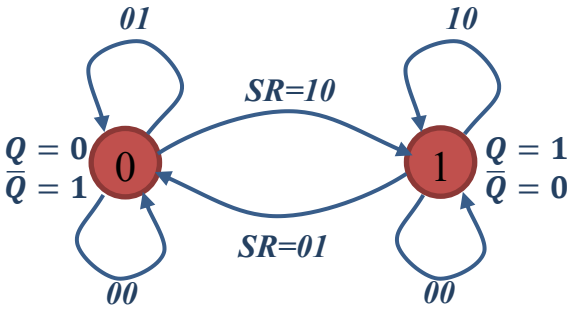
6.4.1 Fizyczna realizacja układów synchronicznych - przerzutniki

W układach synchronicznych pamięć realizowana jest przez tzw. automaty elementarne, które mogą mieć różną budowę i stopień złożoności, ale podstawową ich częścią jest zawsze **przerzutnik.**

W układach elektronicznych podstawową strukturą jest przerzutnik **RS** (ang. RS flip-flop), zbudowany z dwóch bramek logicznych (NAND lub NOR), które są wzajemnie ze sobą sprzężone. Ten przerzutnik ma dwa wejścia: ustawiające S (ang. set) i zerujące R (ang. reset). Standardowo przerzutniki mają również dwa wyjścia komplementarne, tzn. takie, które zawsze mają odwrotne wartości. Dlatego zazwyczaj oznaczane są jako Q i $\bar{Q}$. Schemat przerzutnika RS oraz odpowiadająca mu tabela stanów i graf przejść przedstawia na rys. 6.17.



a)



b)

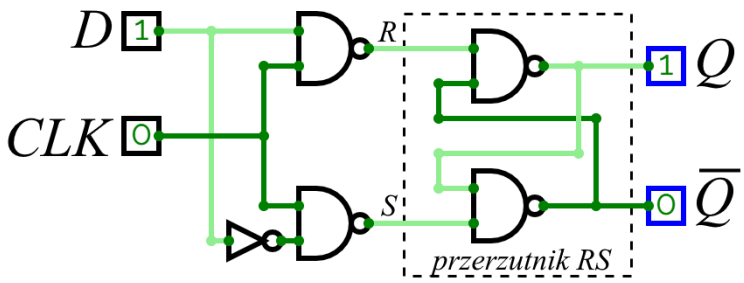
S	R	Q	$\bar{Q}$
0	0	bez zmian	
0	1	0	1
1	1	-	-
1	0	1	0

c)

Rys. 6.17 Przerzutnik RS zbudowany z dwóch bramek NOR (a), graf przejść (b) oraz odpowiadająca tablica stanów (c).

Zgodnie z tablicą przejść z rys. 6.17c przerzutnik zmienia stan tylko gdy $S = \bar{R}$. Przy $SR = 00$ przerzutnik pozostaje w aktualnym stanie, tzn. $Q^{t+1} = Q^t$. Podanie na obu wejściach stanu wysokiego ($SR = 11$) spowodowałoby pojawienie się stanu niskiego na obu wyjściach, co jest niezgodne z przyjętą zasadą komplementarności wyjść. Dlatego ten stan nazywa się niedozwolonym lub zabronionym. Dla przerzutnika RS zbudowanego z dwóch bramek NAND stanem zabronionym jest $SR = 00$, a zmian wyjść nie powoduje stan $SR = 11$.

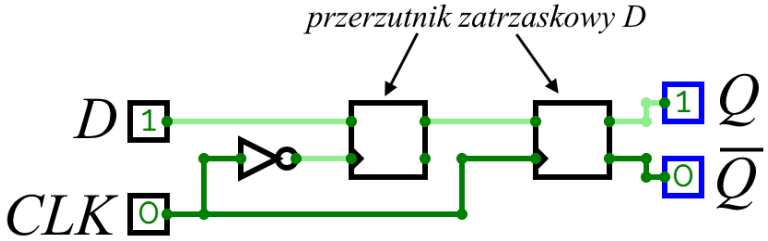
Przerzutnik RS stanowi podstawową strukturę, na której podstawie zbudowane są inne, bardziej skomplikowane przerzutniki – D, T i JK. Przerzutnik RS jest przerzutnikiem asynchronicznym oraz posiada zabroniony stan wejść. Obie te kwestie rozwiązuje synchroniczny **przerzutnik zatrzaskowy D** (ang. D latch), którego schemat pokazano na rys. 6.18.



Rys. 6.18 Przerzutnik zatrzaskowy D ze strukturą przerzutnika RS zbudowaną z bramek NAND.

Ten przerzutnik ma dwa wejścia: ustawiające D i zegarowe CLK. Problem zabronionego stanu w przerzutniku RS jest tutaj rozwiązany przez podawanie stanu D jako sygnał R i sygnału $\overline{D}$ na wejście S. Dzięki zastosowaniu dwóch bramek NAND zmianę stanu wyjść można uzyskać tylko, kiedy sygnał zegarowy ma stan wysoki. Działanie tego przerzutnika można podsumować tak, że stan wejścia D jest przepisywany na wyjście Q, tylko kiedy sygnał zegarowy jest w stanie wysokim.

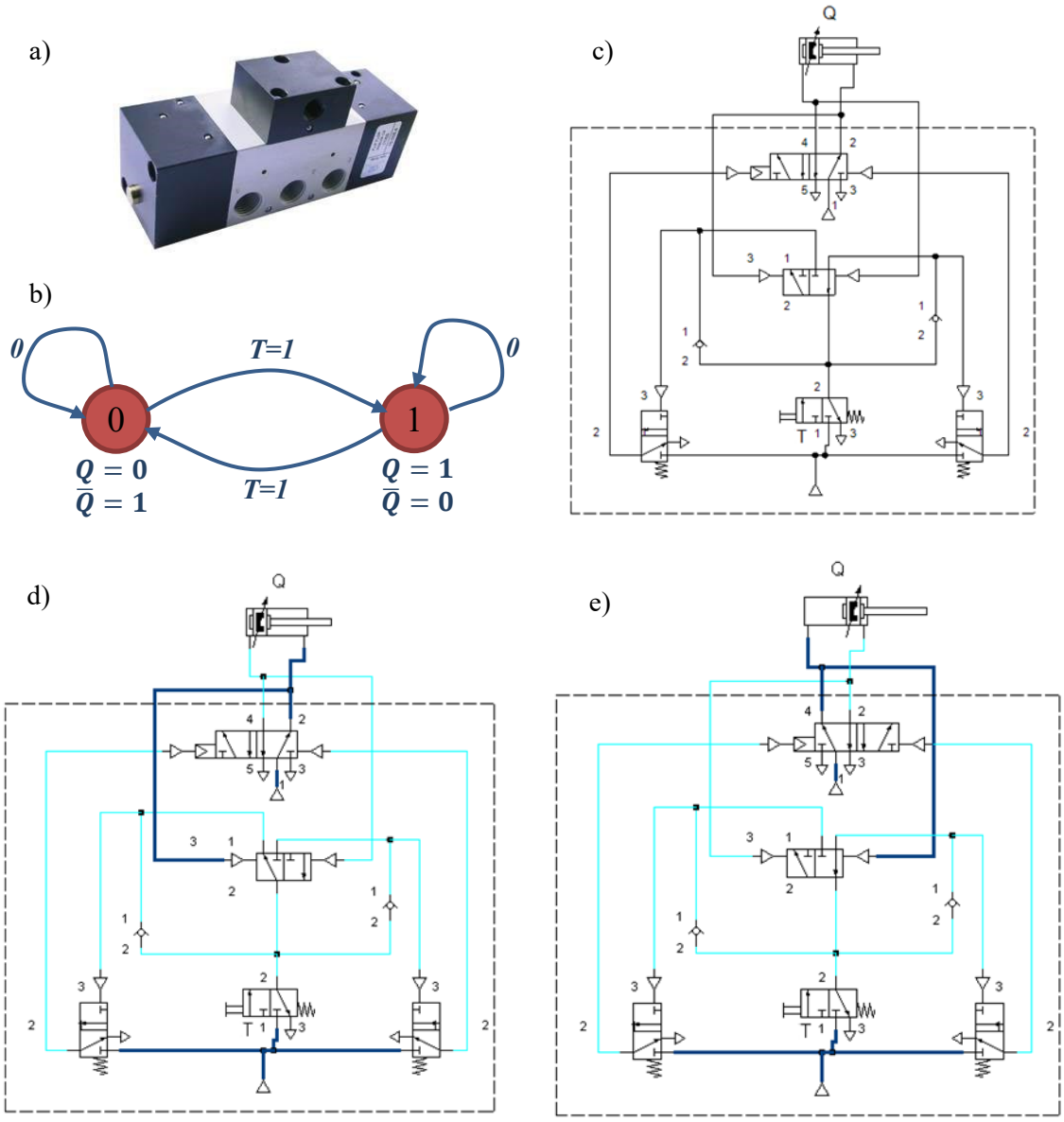
Łącząc dwa przerzutniki zatrzaskowe D w tzw. konfiguracji „master-slave" uzyskuje się **przerzutnik D** (ang. D flip-flop), który przepisuje stan D na wyjście Q, tylko podczas **zbocza narastającego** sygnału zegarowego CLK. Schemat takiego przerzutnika pokazano na rys. 6.19.



Rys. 6.19 Przerzutnik D.

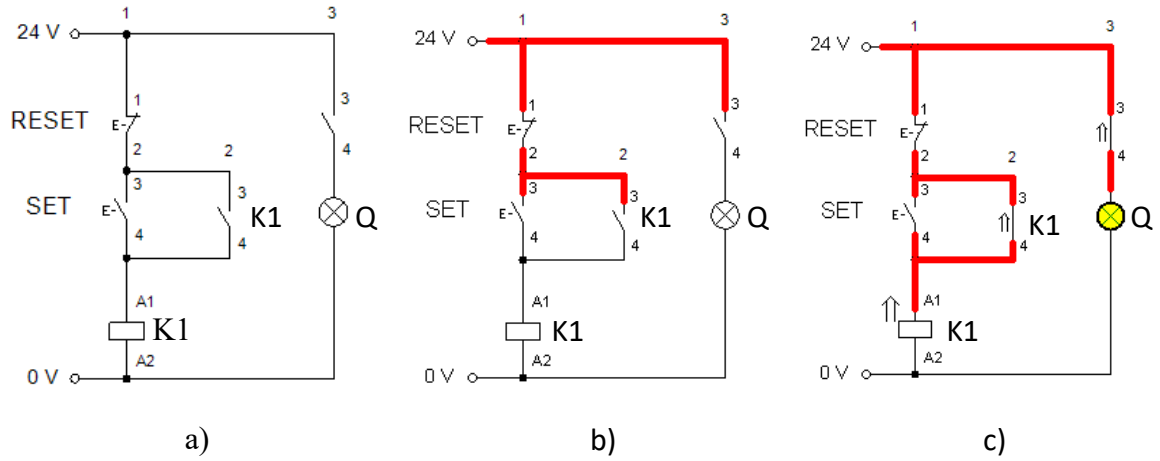
Przerzutniki są elementami występującymi nie tylko w elektronice pod postacią układów scalonych. Wykorzystywane i realizowane są również w innych dziedzinach techniki np. jako zawory pneumatyczne bądź hydrauliczne, elektryczne układy przekaźnikowe, tranzystorowe układy elektroniczne, układy mechaniczne, czy fragmenty programu w sterownikach PLC.

Na rys. 6.20 pokazany jest przerzutnik pneumatyczny. Składa się on z czterech zaworów 3/2, jednego zaworu 5/2 oraz dwóch zaworów zwrotnych. Chwilowe przesterowanie zaworu nazwanego T poprzez wciśnięcie przycisku spowoduje przesterowanie reszty zaworów i wysunięcie siłownika ($Q = 1$). Ponowne chwilowe wciśnięcie przycisku spowoduje jego wsunięcie ($Q = 0$). W układach elektronicznych analogicznie działa przerzutnik T.



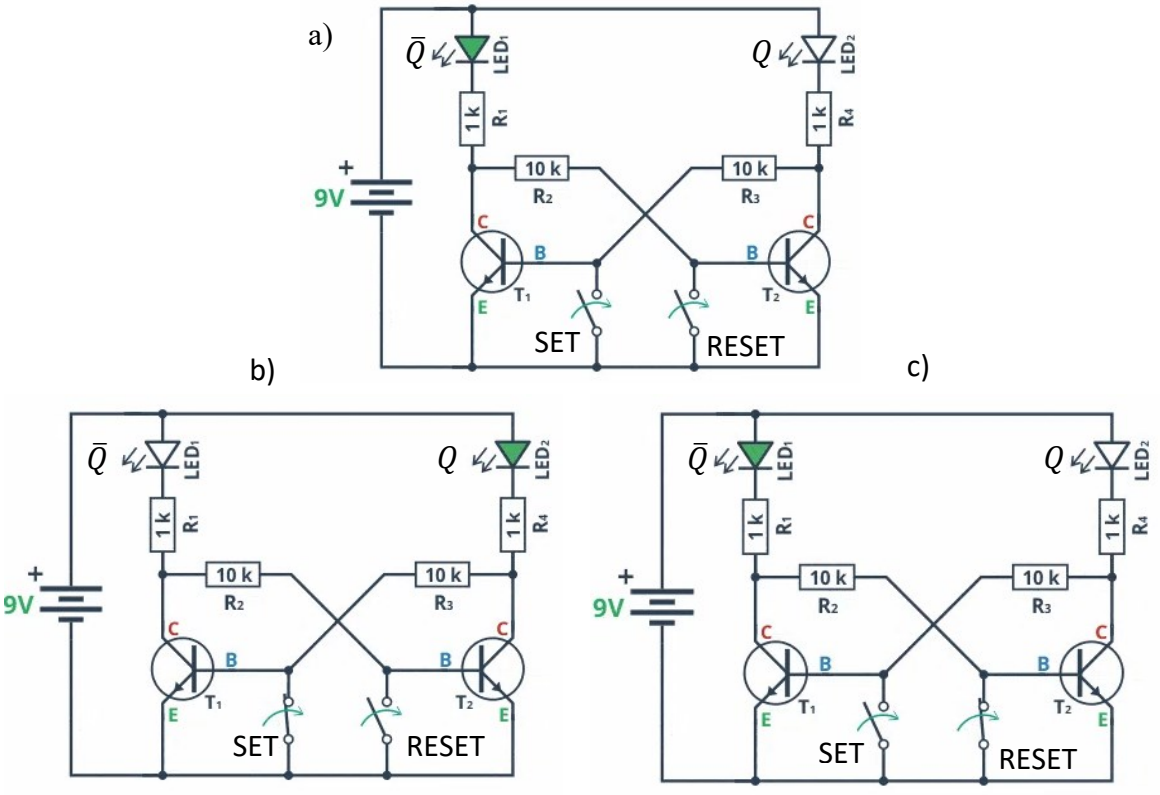
Rys. 6.20. a) rzeczywisty wygląd zaworu pneumatycznego przerzutnika [https://www.soteks-hop.com/en/prod/az-pneumatica/10-035-4/]; b) graf przejść przerzutnika T; c) schemat pneumatyczny przerzutnika; d) przerzutnik w stanie $Q = 0$; e) przerzutnik w stanie $Q = 1$.

Realizację przerzutnika RS z wykorzystaniem elementów elektrycznych – przekaźników, prezentuje rys. 6.21. Układ składa się z dwóch monostabilnych przycisków (SET, RESET), przekaźnika K1 o styku normalnie otwartym (K1) i żarówki (Q). Przycisk normalnie zamknięty pełni rolę RESET, a przycisk normalnie otwarty rolę SET. Wciśnięcie przycisku SET spowoduje zasilenie cewki przekaźnika K1, który zewrze swój styk normalnie otwarty K1. Przez zwarty styk K1 popłynie prąd, który podtrzyma zasilanie cewki przekaźnika nawet po zwolnieniu przycisku SET. Wyjście tego przerzutnika jest tutaj reprezentowane przez żarówkę (Q).



Rys. 6.21. a) schemat przerzutnika elektrycznego na przekaźniku; b) przerzutnik w stanie $Q = 0$; c) przerzutnik w stanie $Q = 1$.

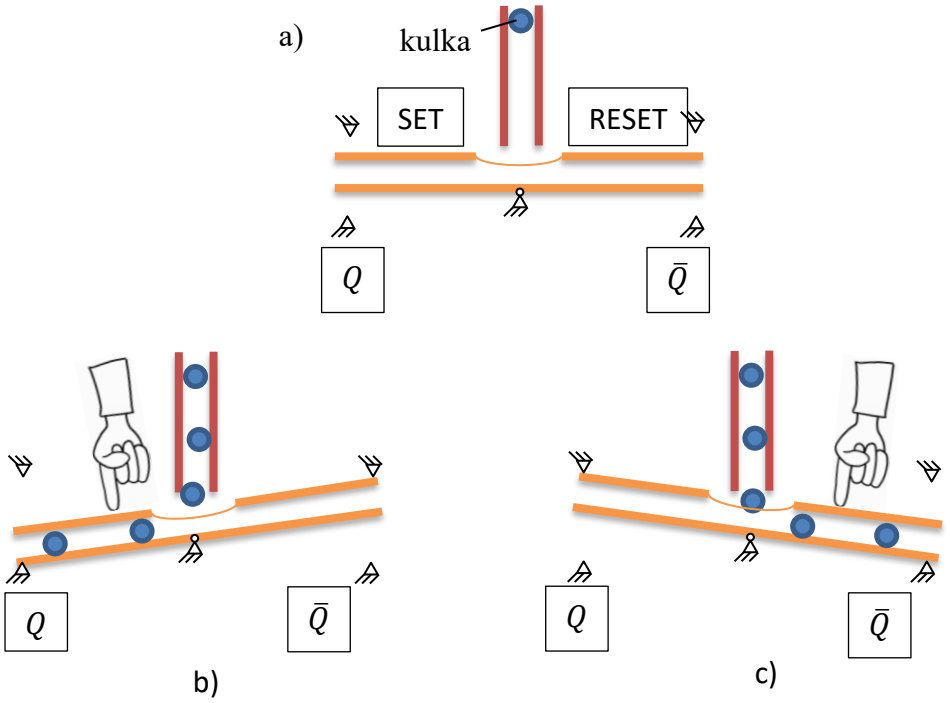
Rys. 6.22 przedstawia układ przerzutnika tranzystorowego, który składa się z dwóch tranzystorów npn, dwóch przycisków monostabilnych pełniących rolę SET, RESET oraz diod LED pełniących rolę wyjść Q. Chwilowe przyciskanie przycisków powoduje załączanie i wyłączanie diod.



Rys. 6.22. a) schemat przerzutnika tranzystorowego; b) przerzutnik w stanie $Q = 1$; c) przerzutnik w stanie $Q = 0$ [https://forbot.pl/blog/kurs-elektroniki-projekty-z-tranzystorami-mosfety-id30808].

Przerzutnik RS może być również zrealizowany jako układ mechaniczny, chociaż możliwości praktycznego zastosowania takiej konstrukcji są raczej małe. Przykładowym przerzutnikiem mechanicznym jest rurka transportująca kulki działajaca na zasadzie dźwigni (patrz Rys. 6.23). Wejściem do układu jest siła przyłożoan w odpowiednim miejscu dźwigni SET lub RESET. Popychajac dźwignię w jednym kierunku przekierowujemy strumień spadających kulek. Po

ustąpieniu działania siły, raz przestawiona przez nas dźwignia nie może sama powrócić do poprzedniego stanu dopóki ponownie nie przytłożymy do niej siły. Tutaj stan wysoki wyjścia Q reprezentują kulki wypadające z odpowiedniego końca dźwigni.

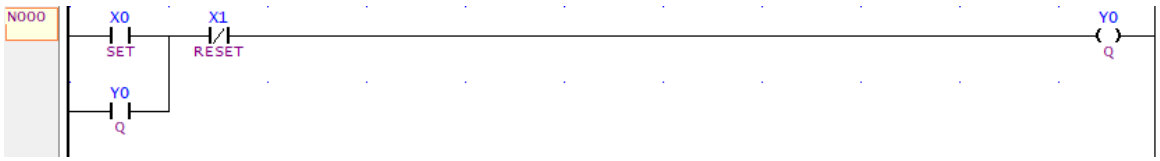


Rys. 6.23. a) przerzutnik dźwigniowy mechaniczny; b) przerzutnik w stanie $Q = 1$; c) przerzutnik w stanie $Q = 0$.

W programach PLC przerzutniki RS mogą być realizowane na kilka sposobów. Zostaną one przedstawione na przykładzie języka drabinkowego LAD. W pierwszym przykładzie (Rys. 6.24a) przerzutnik został zrealizowany analogicznie do układu elektrycznego z rys. 6.21a. Wejściem setującym (ustawiającym) jest X0, które po zwarciu ustawi stan wysoki na wyjściu Y0. Wyjście to załączy swój normalnie otwarty zestyk Y0 i podtrzyma dostarczanie zasilania do wyjścia nawet po zaniku sygnału na wejściu X0. Zresetowanie wyjścia Y0 odbędzie się poprzez przerwanie dostarczania mu zasilania dzięki rozwarciu układu zestykiem resetującym (zerującym) X1. Wciśnięcie w jednej chwili zestyku X0 i X1 spowoduje zresetowanie wyjścia Y0, dlatego przerzutnik ten nazywa się „przerzutnikiem z przewagą resetu”.

Drugi przykład (Rys. 6.24b) przedstawia zrealizowanie przerzutnika RS przy wykorzystaniu wewnętrznych funkcji podtrzymujących i resetujących stan wyjścia. Funkcje te sygnalizowane są literami R i S znajdującymi się wewnątrz symbolu wyjścia. Wyjście Y0 załączane jest w tym przypadku przez koniunkcję dwóch sygnałów X3 i X4. Po logicznym spełnieniu tych warunków wyjście Y0 jest załączane i stan ten utrzymuje się nawet ustaniu tego warunku. Zresetowanie wyjścia odbywa się natomiast poprzez logiczną alternatywę wejść X5 i X6. W przypadku spełnienia warunków zarówno setowania jak i resetowania wyjście Y0 będzie miało stan niski odpowiadający resetowi. Z tego względu przerzutnik zrealizowany w ten sposób również jest przerzutnikiem z przewaga resetu.

a)



b)

Rys. 6.24. a) standardowa realizacja przerzutnika RS w języku LAD; b) realizacja przerzutnika RS przy pomocy wewnętrznych funkcji podtrzymania i resetowania.

6.4.2 Funkcja wzbudzeń przerzutników

Proces syntezy synchronicznych układów sekwencyjnych jest analogiczny do syntezy układów asynchronicznych, z dwiema zasadniczymi różnicami. Analiza stanów stabilnych i ryzyka hazardu statycznego nie jest konieczna, ponieważ potencjalne problemy są rozwiązywane przez synchronizację chwil czasowych zmian stanów układu. Kolejna różnica związana jest z zastosowaniem elementów pamięci. Kiedy zostanie już opracowana funkcja przejść, musi ona jeszcze zostać przepisana na tzw. **funkcję wzbudzeń.** Postać tej funkcji będzie zależała od wybranego elementu pamięci (np. typu przerzutnika), który będzie wykorzystany w układzie. Funkcję wzbudzeń przygotowuje się w oparciu o tablice przejść przerzutników. Rys. 6.25 przedstawia te tablice dla omówionych wyżej przerzutników RS oraz D jak i dla przerzutników JK i T.

Q^t \ SR^t	00	01	11	10
0	0	0	-	1
1	1	0	-	1
	Q^{t+1}			

a)

Q^t \ D^t	0	1
0	0	1
1	0	1
	Q^{t+1}	

b)

Q^t \ JK^t	00	01	11	10
0	0	0	1	1
1	1	0	0	1
	Q^{t+1}			

c)

Q^t \ T^t	0	1
0	0	1
1	1	0
	Q^{t+1}	

d)

Rys. 6.25 Tablice przejść dla przerzutników SR (a), D (b), JK (c) i T (d). Wykorzystane w tabelach oznaczenia *S, R, D, J, K, T* opisują wejścia poszczególnych przerzutników, natomiast *Q* symbolizuje wyjście przerzutnika.

Z tablicy przejść przerzutnika D (Rys. 6.25b) wynika, że $Q^{t+1} = D$, co znaczy ze wyprowadzona funkcja przejść jest jednocześnie funkcją wzbudzeń dla przerzutnika D. Niestety w przypadku pozostałych przerzutników synteza tej funkcji nie jest już taka prosta. Aby wyznaczyć funkcję wzbudzeń należy przygotować **tablicę wzbudzeń**. Taka tablica ma współrzędne takie jak tablica przejść, ale wewnątrz zamiast stanów układu $\boldsymbol{Q^{t+1}}$, będą sygnały wejściowe (ustawiające) przerzutniki -(S, R), D, (J, K) lub T. W opracowaniu funkcji wzbudzeń pomocna może być tablica przerzutników z Rys. 6.26, która podsumowuje przedstawione wyżej tablice przejść poszczególnych przerzutników.

$Q^t \rightarrow Q_1^{t+1}$	D	T	S,R	J,K
0→0	0	0	0,-	0,-
0→1	1	1	1,0	1,-
1→0	0	1	0,1	-,1
1→1	1	0	-,0	-,0

Rys. 6.26 Tablica wzbudzeń przerzutników

Przygotowanie tablicy wzbudzeń zostało zilustrowane na przykładzie z rys. 6.27, gdzie dana jest tablica przejść automatu (Rys. 6.27a), który ma jedno wejście x oraz stany wewnętrzne opisane przez dwie zmienne (q_1, q_2). Rys. 6.27b i Rys. 6.27c. przedstawia tablice wzbudzeń dla przerzutnika T, gdzie T_1 jest zmienna ustawiająca przerzutnik zapamiętujący q_1, natomiast T_2 jest zmienna ustawiającą skojarzoną z q_2.

$(q_1, q_2)^t$ \ x^t	0	1
00	01	00
01	11	00
11	11	00
10	--	--
	$(q_1, q_2)^{t+1}$	

a)

$(q_1, q_2)^t$ \ x^t	0	1
00	0	0
01	1	0
11	0	1
10	-	-
	T_1	

b)

$(q_1, q_2)^t$ \ x^t	0	1
00	1	0
01	0	1
11	0	1
10	-	-
	T_2	

c)

Rys. 6.27 Tablica przejść układu sekwencyjnego (a) oraz odpowiadające jej tablice wzbudzeń dla przerzutnika T (b) i (c). Zmienna T_1 jest skojarzona ze zmienna q_1, a T_2 odpowiada q_2.

Przygotowując tablicę wzbudzeń należy przeanalizować każde przejście $q^t \rightarrow q^{t+1}$ i sprawdzić odpowiadającą mu wartość w tablicach przejść dla przerzutników (Rys. 6.25 lub Rys. 6.26). Przykładowo w tablicy z Rys. 6.27a dla $x = 0$, jest przejście (00) →(01), tzn. dla q_1 (0) →(0), czemu odpowiada $T = 0$, a dla q_2 mamy (0) →(1), czemu odpowiada $T = 1$. Analogicznie wypełniamy resztę tablicy. Jak można zauważyć, zastosowanie przerzutników SR lub JK wiąże się z dwukrotnym zwiększeniem liczby wyjść bloku przejść (dla zmiennej q_1 należałoby przygotować tablice wzbudzeń dla S_1 i R_1, a dla q_2 odpowiednio S_2 i R_2).

6.4.3 Przykład syntezy układu synchronicznego w strukturze Mealy'ego

Synteza synchronicznego automatu w strukturze Mealy'ego zostanie omówiona na przykładzie układu realizującego dodawanie liczb binarnych, tzw. sumatora szeregowego. Krótkie przypomnienie dotyczące dodawania liczb binarnych przedstawia rys. 6.28.

$$\begin{array}{cccccc} & 2^4 & 2^3 & 2^2 & 2^1 & 2^0 \\ & {\scriptstyle 1} & {\scriptstyle 1} & & & \\ & & 1 & 1 & 0 & 1 \\ + & & 1 & 1 & 0 & 0 \\ \hline & 1 & 1 & 0 & 0 & 1 \end{array} \qquad \begin{array}{r} 8+4+0+1= \ 13 \\ 8+4+0+0= \ 12 \\ \hline 16+8+0+0+1= \ 25 \end{array}$$

Rys. 6.28 Przykład dodawani dwóch liczb binarnych.

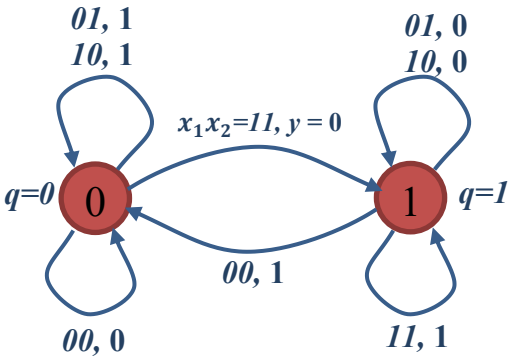
Sumator szeregowy jest układem sumującym dwie liczb binarne, wykonując dodawanie w kolejnych rzędach wielkości (2^0, 2^1, 2^2…itd.). Tak samo jak przy ręcznym dodawaniu „pod kreskę", czasem zachodzi potrzeba przeniesienia wartości „w pamięci" na pozycję bardziej znaczącą. W tym przypadku liczba stanów wewnętrznych (pojemność pamięci) jest znana z góry, ponieważ do przeniesienia może być tylko zero albo jeden. Na rys. 6.29 przedstawiono tablicę przejść, wyjść oraz graf dla automatu Mealy'ego. Projektowany układ ma dwa wejścia $\boldsymbol{X} = (x_1, x_2)$, jedno wyjście $\boldsymbol{Y} = (y)$ i jedną zmienną pamięci $\boldsymbol{Q} = (q)$.

q^t \ $(x_1,x_2)^t$	00	01	11	10
0	0	0	1	0
1	0	1	1	1
	q^{t+1}			

a)

q^t \ $(x_1,x_2)^t$	00	01	11	10
0	0	1	0	1
1	1	0	1	0
	y^t			

b)



c)

Rys. 6.29 Sumator szeregowy w układzie Mealy'ego: tablica przejść (a), tablica wyjść (b) i graf układu (c).

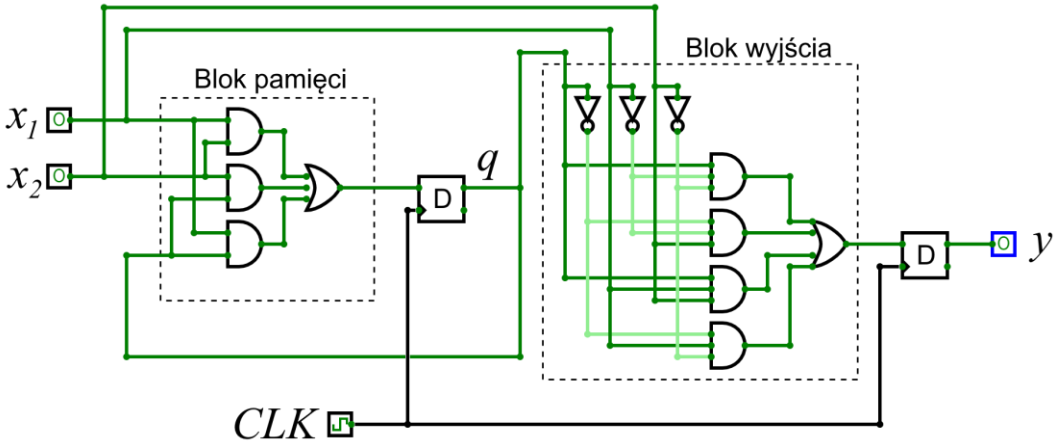
Następnym etapem w syntezie automatu jest redukcja stanów wewnętrznych, jednak w tym przypadku nie znajdziemy już stanów zgodnych. Na podstawie powyższych tablic wyprowadzono następującą funkcję przejść:

$$q^{t+1} = x_1x_2 + qx_2 + qx_1, \tag{6.10}$$

oraz funkcję wyjść:

$$y = q\bar{x}_1\bar{x}_2 + \bar{q}\bar{x}_1x_2 + qx_1x_2 + \bar{q}x_1\bar{x}_2. \tag{6.11}$$

Sumator szeregowy jest typowym układem synchronicznym, ponieważ występuje w nim konieczność odróżniania sąsiednich jednakowych stanów wejść (np. 1101+1100). Następnym krokiem jest wybór przerzutników, które zostaną wykorzystane w bloku pamięci oraz synteza funkcji wzbudzeń. Jeżeli wybierzemy przerzutniki D (które po prostu przepisują stan wejścia D na wyjście Q) możemy od razu przystąpić do budowy układu, ponieważ funkcja wzbudzeń jest tożsama z funkcją przejść. Układ logiczny zaprojektowany na bramkach logicznych na podstawie równań (6.10) i (6.11) przedstawia rys. 6.30.



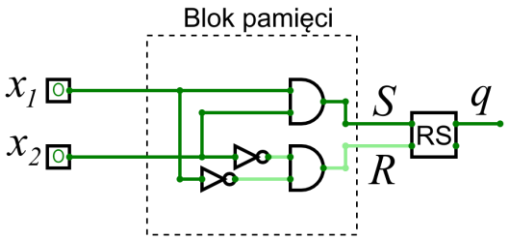
Rys. 6.30 Sumator szeregowy w układzie Mealy'ego z przerzutnikami D.

Projektując synchroniczny automat Mealy'ego należy rozwiązać problem **asynchronicznych zmian stanu wyjścia**. W automacie Moore'a wyjście zależy tylko od stanu pamięci, dlatego będzie się ono zmieniało synchronicznie. W automacie Mealy'ego wyjście układu jest obliczane zarówno na podstawie stanu wejść jak i stanu pamięci, co może powodować chwilowo błędne stany wyjść. Rozwiązanie tego problemu ogólnie zależy od danego przypadku oraz współpracujących układów logicznych „podających wejścia" i „odbierających wyjścia". W układzie z rys. 6.30 problem ten rozwiązano stosując dodatkowy przerzutnik D przed wyjściem układu synchronizowany tym samym sygnałem zegarowym CLK, co przerzutnik zmiennej pamięci.

Opracowując powyższy układ z wykorzystaniem innego przerzutnika niż D, należy przygotować funkcje wzbudzeń. Poniżej (Rys. 6.31a i Rys. 6.31b) przedstawiono tablice wzbudzeń dla przerzutnika RS opracowane na podstawie tablicy przejść z rys. 6.29a oraz tablicy z rys. 6.26.

q^t \ $(x_1,x_2)^t$	00	01	11	10
0	0	0	1	0
1	0	-	-	-
	S			

q^t \ $(x_1,x_2)^t$	00	01	11	10
0	-	-	0	-
1	1	0	0	0
	R			



a) b) c)

Rys. 6.31 a) tablica wzbudzeń dla zmiennej S; b) tablica wzbudzeń dla zmiennej R; c) blok pamięci dla przerzutnika RS.

Na podstawie powyższych tablic wyprowadzono następujące funkcje wzbudzeń przerzutnika RS:

$$S = x_1 x_2, \tag{6.12}$$

$$R = \bar{x}_1 \bar{x}_2. \tag{6.13}$$

Wyprowadzonym funkcjom odpowiada układ logiczny przedstawiony na rys. 6.31c. Wykorzystanie przerzutnika RS wymaga zwiększenia liczby wyjść z bloku pamięci, ale z drugiej strony spowodowało uproszczenie jego wewnętrznej struktury.

7 Literatura

- Praca zbiorowa pod red. Mirosława Roszkowskiego, *Laboratorium teorii maszyn, drgań mechanicznych i podstaw automatyki*, Wydawnictwo Politechniki Łódzkiej, Łódź 1988.
- W. Traczyk, *Układy cyfrowe automatyki*, Wydawnictwo Naukowo-Techniczne, Warszawa 1974.
- J. Kalisz, *Podstawy elektroniki cyfrowej*, Wydawnictwo Komunikacji i Łączności, Warszawa 2002.
- T. Łuba, *Syneza układów logicznych,* Oficyna Wydawnicza Politechniki Warszawskiej, Warszawa 2005.
- Dokumentacja symulatora układów logicznych CircuitVerse, *https://circuitverse.org/*