Podstawy techniki cyfrowej
zima 2015

Wykład

dr inż. Rafał Walkowiak

Literatura

1. Podstawy Techniki Cyfrowej, Barry Wilkinson, WKiŁ 2001
2. Podstawy Projektowania Układów Cyfrowych, Cezary Zieliński, PWN 2012
3. Fundamentals of computer engineering - Logic design and microprocessors, H.Lam, J. O, J. Wiley and Sons, 1998
4. Język VHDL: projektowanie programowalnych układów logicznych, Kevin Shakill, WNT 2004
5. Układy cyfrowe, Zbiór zadań z rozwiązaniami, J.Tyszer, G.Mrugalski, Wydawnictwo PP

6. Układy Scalone TTL w systemach cyfrowych, J. Pienkos, J. Turczyński, Wkił, 1994
7. Cyfrowe układy scalone MOS, P. Gajewski, J.Turczyński, WKiŁ, 1998

Zakres przedmiotu

- Wstęp: arytmetyka binarna, algebra Boole'a , kody binarne, BCD, podstawowe funkcje logiczne, sposoby przedstawiania funkcji logicznych - postacie kanoniczne, minimalizacja funkcji logicznych, łączna minimalizacja funkcji logicznych, hazard.
- Technologie CMOS,TTL i ich wpływ na właściwości użytkowe układów, bramki logiczne.
- Układy kombinacyjne: multipleksery i demultipleksery; komparatory, łączenie komparatorów; kodery, dekodery, translatory kodów; sumatory: sumatory binarne, dziesiętne.
- Podstawowe elementy sekwencyjne: zatrzask RS, zatrzask D, przerzutniki: D, JK, T; parametry czasowe, rejestry szeregowe, równoległe, przesuwne, rejestry liczące.
- Liczniki: synchroniczne i asynchroniczne, binarne, dziesiętne; łączenie liczników, synteza liczników, skracanie liczników, taktowanie systemów cyfrowych, częstotliwości maksymalne liczników;
- Automaty synchroniczne: Moora, Mealego, graf i tablica przejść automatu, minimalizacja stanów, kodowanie stanów, funkcje przejść i wyjść i implementacja automatu na przerzutnikach.
- Język opisu sprzętu VHDL : jednostki projektowe, obiekty, typy, typy rozstrzygalne, instrukcje współbieżne i sekwencyjne, komponenty, strukturalny i behawioralny opis układów, przykładowe realizacje układów kombinacyjnych, sekwencyjnych, automatów.
- Układy programowalne: ROM, PLD, PLA, PAL, FPGA.
- Synteza wyższego poziomu: implementacja układów cyfrowych dla realizacji algorytmów przetwarzania danych; , opisy układu: sieć działań algorytmu, diagram synchronicznego układu sekwencyjnego, diagram synchronicznego układu sekwencyjnego ze zintegrowaną ścieżką danych; projekt: schemat strukturalny, opis układu cyfrowego w języku opisu sprzętu.
- Układy mikroprogramowalne w sterowaniu układami cyfrowymi.
- Pamięci: statyczne i dynamiczne, RAM, CAM, łączenie pamięci, parametry, cykle zapisu i odczytu.
- Współpraca układów cyfrowych z otoczeniem; wprowadzanie i wyprowadzanie danych, wyświetlanie statyczne i dynamiczne.
- Sposoby organizacji systemów cyfrowych: iteracja w czasie i przestrzeni.
- Automaty asynchroniczne, minimalizacja liczby stanów i kodowanie stanów, przykłady implementacji.

Systemy cyfrowe

- System cyfrowy – to układ powiązanych ze sobą elementów projektowany w celu realizacji takich zadań jak:
  - przetwarzanie informacji (w tym obliczenia)
  - sterowanie urządzeniami i innymi systemami i obiektami (np. silniki, zawory, piece itp.)
- Przetwarzane informacje zapisane są za pomocą wartości z określonego ograniczonego zbioru (np. cyfr w różnych (dla wygody) systemach liczenia).

Systemy liczenia *

- Co już wiemy: [1, str 15-22]
  - Pozycyjne systemy liczenia – dziesiętny, dwójkowy, ósemkowy, szesnastkowy
  - Konwersje liczb między systemami, konwersje liczb ułamkowych
- Systemy uzupełnieniowe:
  - **Uzupełnienie do K (do podstawy K)**- **Uzupełnienie liczby N** zapisanej w systemie o podstawie K do K (podstawy K) definiujemy:
  - $K^n - N$
  - Gdzie n jest liczbą cyfr liczby N

  Np. U10(345)=655 U2(10101)=001011
  - **Uzupełnienie do K -1 (do podstawy K -1)**- Uzupełnienie liczby N zapisanej w systemie o podstawie K do (podstawy) K definiujemy:
  - $K^n - 1 - N$
  - 345_{10} U9(344)=655= U10(345)=655 10101_2 U1(1010)=U2(1011)

* Literatura: Wilkinson, Strony 15-28

Reprezentacje liczb ze znakiem

- Znak moduł – najstarszy bit określa znak liczby, pozostałe bity bez zmiany
- Zastosowanie kodu U2 – bit znaku i moduł liczby ujemnej w kodzie U2
- Zastosowanie kodu U1 – bit znaku i moduł liczby ujemnej w kodzie U1

N	ZM	U2	U1	N	ZM
-8	-	1000	-		
-7	1111	1001	1000	7	0111
-6	1110	1010	1001	6	0110
-5	1101	1011	1010	5	0101
-4	1100	1100	1011	4	0100
-3	1011	1101	1100	3	0011
-2	1010	1110	1101	2	0010
-1	1001	1111	1110	1	0001
				0	0000

Reprezentacja uzupełnieniowa

Do zapisu liczb ujemnych – użycie kodu U2

- Binarna liczba **dodatnia** jest zapisywana na wystarczającej liczbie pozycji i uzupełniana zerami na pozycjach bardziej znaczących: $(3)_{10} = (011)_2 = (0011)_2$
- Binarna liczba **ujemna** jest zapisywana:
  - w uzupełnieniu do 2 i
  - poprzedzona 1 na pozycji najstarszej i
  - uzupełniona jedynkami na pozycjach bardziej znaczących: $(-3)_{10} = (101)_2 = (1101)_2$
- Notacja uzupełnieniowa liczb binarnych pozwala na dodawanie liczb dodatnich i ujemnych (realizowane standardowo jak dla liczb binarnych w NKB - sumator).

Dodawanie liczb ujemnych wykorzystanie notacji U2

1101	-3
1110	+(-2)
(1)1011	= -5

1101	-3
0010	+2
1111	= -1

1101	-3
0101	+5
(1)0010	= 2

Przeniesienie ignorowane, przeniesienia na najstarszy bit i z najstarszego bitu są jednakowe.

Odejmowanie liczb – dodawanie liczby przeciwnej

```
   0011 (3d)
+  1011 (-5d)
-------------
   1110 (-2d)

   0101 (5d)
+  1101 (-3d)
---------------
   0010  (2d)
```

Binarna liczba ujemna – liczba binarna w uzupełnieniu do 2

00010110 = 22 (d)

Wyznaczenie liczby U2 –

Medtoda 1:

11101001 negacja bitów

11101010 dodanie jedynki = -22 (d)

Metoda 2:

Negacja bitów bardziej znaczących - starszych niż najmniej znaczący bit równy 1.

Odejmowanie binarne

- D – dodatnia
- U – ujemna
- D – U= D+D=D (sprawdzenie przepełnienia)
- D1 – D2 = D gdy (D1>D2) lub U gdy (D1<D2)
- U-D= U + U = U (sprawdzenie przepełnienia)

Dodawanie liczb a przepełnienie

```
   0011 (3d)
+  0011 (3d)
------------
   0110 (6d) wynik dodatni – poprawnie
```

```
   1101 (-3)
+  1101 (-3)
----------------
   1010 (-6) wynik ujemny – poprawnie
```

```
   0101 (5d)
+  0101 (5d)
---------------
   1010  -(6d) Wynik ujemny - niepoprawny
```

```
   1011 (-5)
+  1011 (-5)
-----------------
   0110 (6) wynik dodatni - niepoprawny
```

Wynik niepoprawny – przepełnienie – nadmiar - gdy przeniesienia na najwyższą pozycję i z najwyższej pozycji są różne.

Kody dwójkowo-dziesiętne

- 10 cyfr dziesiętnych (0,1,2,3,4,5,6,7,8,9) zakodowanych za pomocą ciągu 4 bitów – 6 kombinacji (z 16) tych 4 bitów jest niewykorzystanych.

- Kody wagowe – pozycja binarna posiada przypisaną wagę
- Kody niewagowe – pozycja binarna nie posiada wagi

Kody dwójkowo-dziesiętne wagowe

kod	Naturalny NKB		Aikena		
Wagi cyfra	8421	2*421	2421	7421	84-2-1
0	0000	0000	0000	0000	0000
1	0001	0001	0001	0001	0111
2	0010	0010	0010	0010	0110
3	0011	0011	0011	0011	0101
4	0100	0100	0100	0100	0100
5	0101	0101	1011	0101	1011
6	0110	0110	1100	0110	1010
7	0111	0111	1101	1000	1001
8	1000	1110	1110	1001	1000
9	1001	1111	1111	1010	1111

Kody dwójkowe niewagowe

Kod cyfra	Z nadmiarem 3	Graya	Wattsa	Johnsona	Wskaźników 7 segmentowych	
0	0011	0000	0000	00000	0111111	7
1	0100	0001	0001	00001	0000110	2 6
2	0101	0011	0011	00011	1011011	1
3	0110	0010	0010	00111	1001111	3 5
4	0111	0110	0110	01111	1100110	4
5	1000	0111	1110	11111	1101101	
6	1001	0101	1010	11110	1111100	
7	1010	0100	1011	11100	0000111	
8	1011	1100	1001	11000	1111111	
9	1100	1101	1000	10000	1100111	

Kody detekcyjne

kod	1 z 10	2 z 5	2 z 7	Bin z Bitem parzystości
Wagi cyfra	9876543210	niewagowy	5043210	8421 0
0	0000000001	00011	0100001	0000 0
1	0000000010	00101	0100010	0001 1
2	0000000100	01001	0100100	0010 1
3	0000001000	10001	0101000	0011 0
4	0000010000	00110	0110000	0100 1
5	0000100000	01010	1000001	0101 0
6	0001000000	10010	1000010	0110 0
7	0010000000	01100	1000100	0111 1
8	0100000000	10100	1001000	1000 1
9	1000000000	11000	1010000	1001 0

Kody z kontrolą parzystości i ze stałą liczbą jedynek pozwalają na wykrycie pewnych błędów przy przesyłaniu słów kodowych.

Liczby dziesiętne kodowane dwójkowo – kod BCD 8421

- Dziesiętny charakter informacji lecz kodowanie dwójkowe cyfr
- 2345 (10)= 0010 0011 0100 0101(BCD)
- Dodawanie liczb w kodzie BCD realizowane tak jak dodawanie liczb binarnych, lecz:
  - wystąpienie przeniesienia na kolejną pozycję dziesiętną (kolejne 4 bity) podczas dodawania liczb wymaga skorygowania (czyli dodania wartości 6) na tej pozycji, z której przeniesienie wystąpiło
  - wystąpienie wyniku na 4 bitach (pozycji dziesiętnej) spoza zakresu (10-15) wymaga skorygowania wyniku czyli dodania wartości 6 na tej pozycji dziesiętnej, która nie jest poprawna, (może wystąpić przeniesienie, które należy uwzględnić oraz propagacja przeniesienia np. dla liczb) 3456 +6545.

Dodawanie w kodzie BCD

```
 89              1000 1001
+18         +    0001 1000
-----           ----------------------
107             1010 0001 przeniesienie
                     0110
                ----------------------
                1010 0111 wartość spoza przedziału
                0110
                ----------------------
               1 0000 0111
```

Kody alfanumeryczne

- Kody służące do kodowania znaków w systemach cyfrowych, w urządzeniach współpracujących z komputerem, np. drukarki, ekrany alfanumeryczne.
- Przykładami kodów alfanumerycznych są kody: ASCII ISO-7, ISO 8859, Unicode, Windows-1250.
- Kod ASCII – ISO-7 7 bitowy – pełny zbiór zawiera 128 znaków, pierwsze 33 znaki służą do sterowania systemu drukowania lub wyświetlania, pozostałe znaki to: duże i małe litery, cyfry, znaki przestankowe i inne.

Kod ISO-7

Kod ISO-7 Tablica 2.11

	b_7	0	0	0	0	1	1	1	1
	b_6	0	0	1	1	0	0	1	1
	b_5	0	1	0	1	0	1	0	1
b_4 b_3 b_2 b_1									
0 0 0 0		NUL	DLE	SPACE[2]	0[2]	@[1]	P	\[1]	p
0 0 0 1		SOH	DC1	!	1	A	Q	a	q
0 0 1 0		STX	DC2	„	2	B	R	b	r
0 0 1 1		ETX	DC3	≠	3	C	S	c	s
0 1 0 0		EOT	STOP	S[1]	4	D	T	d	t
0 1 0 1		ENQ	NAK	%	5	E	U	e	u
0 1 1 0		ACK	SYN	&	6	F	V	f	v
0 1 1 1		BEL	ETB	,	7	G	W	g	w
1 0 0 0		BS	CAN	(	8	H	X	h	x
1 0 0 1		HT	EM	)	9	I	Y	i	y
1 0 1 0		LF	SUB	*	:	J	Z	j	z
1 0 1 1		VT	ESC	+	;	K	[1]	k	{[1]
1 1 0 0		FF	FS	”	<	L	\[1]	l	|[1]
1 1 0 1		CR	GS	—	=	M	]	m	}
1 1 1 0		SO	RS	.	>	N	∧[1]	n	—[1]
1 1 1 1		SI	US	/	?	O	—	o	DEL

1) Wymienne, 2) Wyjątki dla kodu karty dziurkowanej: literze b odpowiada brak dziurek, cyfrze 0 odpowiada otwór w kolumnie 0

Objaśnienia:

NUL — bez informacji
SOH — początek nagłówka
STX — początek tekstu
ETX — koniec tekstu
EOT — koniec transmisji
EWQ — zapytanie
ACK — odpowiedź pozytywna
BEL — dzwonek
BS — ruch powrotny, cofanie
HT — tabulacja pozioma
LF — zmiana wiersza
VT — tabulacja pionowa
FF — zmiana formularza
CR — powrót wózka
SO — poza kodem
SI — w kodzie

DLE — zmiana znaczenia ciągu znaków
DC1 — sterowanie urządzeniem 1
DC2 — sterowanie urządzeniem 2
DC3 — sterowanie urządzeniem 3
STOP — stop
NAK — odpowiedź negatywna
SYN — synchronizacja
ETB — koniec transmisji bloku danych
CAN — nieważny, anulowanie
EM — koniec zapisu, koniec nośnika informacji
SUB — zastąpienie, podstawienie
ESC — przełączenie, zmiana zestawu znaków
FS — oddzielenie głównych grup
GS — oddzielenie grupy informacji
RS — oddzielenie podgrup (pozycji)
US — oddzielenie części grup
DEL — kasowanie (znak ignorowany)

Algebra Boole'a *

Narzędzie matematyki (algebra logiki) służąca do opisu i projektowania systemów cyfrowych.

Zmienne boolowskie – mogą przyjąć jedna z dwóch wartości – 0 lub 1 – są to zmienne binarne (jednobitowe)

Podstawowe funkcje algebry Boola –

- Iloczyn logiczny I (AND) – „· " ,,∩'' ,,∧'' (alternatywne oznaczenia)
- Suma logiczna LUB (OR) – ,,+'' ,,∪'' ,,∨'' (alternatywne oznaczenia)
- Negacja NIE (NOT) – „linia nad zmienną" ,, ' '' (alternatywne oznaczenia)

Funkcja boolowska (logiczna, przełączajaca) – jest działaniem na zmiennych boolowskich i przyjmuje wartości ze zbioru {0,1}.

Algebra Boole'a jest zgodna z następującymi postulatami:

* Literatura Wilkinson 35-53

Postulaty Huntingtona (1)

Notacja: $Z = \{0,1\}$ – zbiór wartości

a, b – dowolne zmienne binarne

A1 Domknięcie działań: $a + b \subset Z$ $A \cdot B \subset Z$

A2 Elementy stałe: Istnieją takie 0 i 1 : $a+0=a$ i $a \cdot 1=a$

A3 Przemienność: $a+b=b+a$ $a \cdot b= b \cdot a$

A4 Rozdzielność: $a \cdot (b+c)=a \cdot b+a \cdot c$

$a+(b \cdot c)=(a+b) \cdot (a+c)$ również mnożenia względem dodawania

A5 Istnienie negacji: dla a istnieje a’ : $a+a'=1$ $a \cdot a'=0$

Postulaty Huntingtona (2)

Zasada dualności:

Wyrażenie dualne powstanie poprzez zamianę operatorów binarnych i stałych: $+ \rightarrow \cdot$, $\cdot \rightarrow +$, $0 \rightarrow 1$, $1 \rightarrow 0$

Jeżeli prawdziwe jest pewne wyrażenie A to prawdziwe jest również wyrażenie do niego dualne do A.

np wyrażenie proste: $a \cdot (b+c) = a \cdot b + a \cdot c$

Wyrażenie dualne: $a+(b \cdot c) = (a+b) \cdot (a+c)$

Przekształcanie funkcji logicznych

- Dla minimalizacji postaci wyrażeń (funkcji) boolowskich służą tożsamości i twierdzenia algebry boole’a.
- Minimalizacja pozwala na uzyskanie prostszej, tańszej implementacji funkcji.

Twierdzenia algebry Boole'a

- Idempotentność (łac. taki sam) –

  $a+a=a$, $a \cdot a=a$

- Jednoznaczność negacji –

  dla kazdego a istnieje tylko jeden element $\bar{a}$

- Dominacja - dla każdego a $a \cdot 0 = 0$ $a+1=1$
- Podwójna negacja –

  dla kazdego a zachodzi $a = \bar{\bar{a}}$

- Pochłanianie - $a+(a \cdot b)= a$ $a \cdot (a+b)= a$

Twierdzenia algebry Boole'a

- Uproszczenie

$$a + (\bar{a} \cdot b) = a + b \qquad \mathrm{a} \cdot (\bar{\mathrm{a}} + \mathrm{b}) = \mathrm{a} \cdot \mathrm{b}$$

a(1+b)+a'b=a+b(a+a')=a+b

- Minimalizacja - $a \cdot b + a \cdot \bar{b} = a \qquad (a+b) \cdot (a+\bar{b}) = a$
- Łączność – (a+b)+c=a+(b+c) (a·b) ·c=a ·(b ·c)
- Konsensus (zgoda) –

Wystarczy jedna 1,

wystarczy jedno 0

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

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

Prawo de Morgana

$$\overline{a+b+c+\ldots} = \bar{a}\cdot\bar{b}\cdot\bar{c}\cdot\ldots$$

$$\overline{a\cdot b\cdot c\cdot\ldots} = \bar{a}+\bar{b}+\bar{c}+\ldots.$$

Funkcje logiczne dwóch zmiennychi ich wartości

zmienne binarne a b

Wartości	**ab**	**ab**	**ab**	**ab**	**Równanie**	**Nazwa**	**Skrót**
argumentów	00	01	10	11	funkcji	funkcji	Nazwy
Wartości	0	0	0	0	0	Stała Zero	
funkcji	0	0	0	1	a·b	Iloczyn logiczny	AND
	0	0	1	0	a·b’	Zakaz przez b	
	0	0	1	1	a	Identyczna z a	
	0	1	0	0	a’·b	Zakaz przez a	
	0	1	0	1	b	Identyczna z b	
	0	1	1	0	(a’·b)+(a·b’)	Suma modulo	XOR
	0	1	1	1	a+b	Suma logiczna	OR
	1	0	0	0	(a+b)’	Negacja sumy	NOR
	1	0	0	1	(a·b)+(a’·b’)	Równoważność	EQU
	1	0	1	0	b’	Negacja b	
	1	0	1	1	a+b’	Implikacja b $\Rightarrow$a	
	1	1	0	0	a’	Implikacja a	
	1	1	0	1	a’+b	Implikacja a$\Rightarrow$b	
	1	1	1	0	(a·b)’	negacja iloczynu	NAND
	1	1	1	1	1	Stała 1	

Popularne funkcje logiczne

- Szczególnie popularne AND, OR, NAND, NOR,XOR,NOT
- XOR – wartość funkcji równa 1 dla różnych argumentów
- Zależności dla XOR:
  - $a \oplus b = a'b + b'a = (a+b)(a'+b')$
  - $(a \oplus b)' = a' \oplus b = b' \oplus a = ab + a'b' = (a'+b)(a+b')$ XNOR
  - $a \oplus 1 = a'$ $a \oplus 0 = a$
- Różne interpretacje logiczne wielowejściowych bramek XOR/XNOR. Najczęściej bramka wykrywa nieparzystą liczbę jedynek (XOR) lub parzystą liczbę jedynek XNOR.

System funkcjonalnie pełny - SFP

- Zbiór funkcji pozwalający na przedstawienie Wyrażenie każdej innej funkcji logicznej.
- 3 przykłady S.F.P:
  - {NAND},
  - {OR,AND,NOT},
  - {NOR}

Sposoby przedstawiania funkcji logicznych

- Tablica prawdy

j	$x_0 x_1 x_{2***} x_{n-1}$	f
0	0 0 0 *** 0	Wartości funkcji
1	0 0 0 *** 1	
2		
3		
4		
5		
*		
*		
*		
2^n -1	1 1 1 *** 1	

np.

nr	we	wy
0	00	1
1	01	1
2	10	1
3	11	0

- Nr kombinacji wejść, wartości kombinacji wejść, odpowiadające wejściu wartości na wyjściu
- Zawiera wszystkie kombinacje zero-jedynkowe zmiennych niezależnych i odpowiadające im wartości funkcji

Sposoby przedstawiania funkcji logicznych

- Tablice Karnaugha
- Kombinacji wejść odpowiada pole tablicy, w polu umieszczmy właściwą dla kombinacji wartość.
- Sąsiednie (w poziomie i pionie – także cyklicznie) pola tablicy Karnaugha odpowiadają kombinacji argumentów różniącej się jedną wartością.
- Na rysunku zapisano kombinacje wejść – nie wartości

		a	
b		0	1
	0	00	01
	1	10	11

		ba			
c		00	01	11	10
	0	000	001	011	010
	1	100	101	111	110

Tablica dla funkcji 2 i 3 zmiennych wejściowych

Reprezentacja funkcji logicznych za pomocą tablic Karnaugha

b \ a	0	1
0	0	1
1	0	1

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

dc \ ba	00	01	11	10
00	1	1	1	1
01	0	∅	0	0
11	0	∅	0	0
10	1	1	0	1

∅ - oznaczenie wartości dowolnej na wyjściu

Sposoby przedstawiania funkcji logicznych

- Dysjunkcyjna (alternatywna) postać kanoniczna:

$$Y = f(x_0, x_1, \ldots, x_{n-1}) = \bigcup_{j=0}^{2^n-1} a_j I_j$$

- Gdzie: U to suma
- I_j oznacza iloczyn zmiennych niezależnych dla j-tej kombinacji wartości zmiennych równy 1
  - np. zerowa kombinacja: 0000: iloczyn - $x_0‘x_1‘x_2‘x_3‘$ (wartość iloczynu dla kombinacji wartości zmiennych wynosi jeden)
- a_j wartość funkcji odpowiadająca j-tej kombinacji zmiennych
- MINTERM- każda kombinacji argumentów (wejść), dla której wartość funkcji jest równa 1

Dysjunkcyjna postać kanoniczna – przykład

iloczyny	abC_{in}	S	Cout
0 $a'b'c_{in}'$	000	0	0
1 $a'b'c_{in}$	001	1	0
2 $a'bc_{in}'$	010	1	0
3 $a'bc_{in}$	011	0	1
4 $ab'c_{in}'$	100	1	0
5 $ab'c_{in}$	101	0	1
6 abc_{in}'	110	0	1
7 abc_{in}	111	1	1

$S = 0a'b'c_{in}' + 1a'b'c_{in} + 1a'bc_{in}' + 0a'bc_{in} + 1ab'c_{in}' + 0ab'c_{in} + 0abc_{in}' + 1abc_{in}$

$S = a'b'c_{in} + a'bc_{in}' + ab'c_{in}' + abc_{in}$

S = $\cup$(1,2,4,7) – gdzie liczby oznaczają numer kolejny iloczynu (minterm) dla którego wartość funkcji =1

Sposoby przedstawiania funkcji logicznych

- Koniunkcyjna (iloczynowa) postać kanoniczna:

$$Y = f(x_0, x_1, \ldots, x_{n-1}) = \prod_{j=0}^{2^n-1} (a_j + S_j)$$

- Gdzie:
- S_j oznacza sumę zmiennych niezależnych dla j-tej kombinacji zmiennych równą 0
  - Np. kombinacja wejść : 0000; suma dla tej kombinacji: $x_0+x_1+x_2+x_3$,
- a_j oznacza wartość funkcji odpowiadającej j-tej kombinacji zmiennych.
- MAXTERM - każda kombinacji argumentów (wejść), dla której wartość funkcji jest równa 0.

Konjunkcyjna postać kanoniczna - przykład

sumy	abC_{in}	S	Cout
0 $a+b+c_{in}$	000	0	0
1 $a+b+c_{in}'$	001	1	0
2 $a+b'+c_{in}$	010	1	0
3 $a+b'+c_{in}'$	011	0	1
4 $a'+b+c_{in}$	100	1	0
5 $a'+b+c_{in}'$	101	0	1
6 $a'+b'+c_{in}$	110	0	1
7 $a'+b'+c_{in}'$	111	1	1

$$S = (0+ a+b+c_{in})\ (1+a+b+c_{in}')(1+a+b'+c_{in})(0+a+b'+c_{in}')$$
$$(1+a'+b+c_{in})(0+a'+b+c_{in}')(0+a'+b'+c_{in})(1+a'+b'+c_{in}')$$

$$S = (a+b+c_{in})\ (a+b'+c_{in}')(a'+b+c_{in}')(a'+b'+c_{in})$$

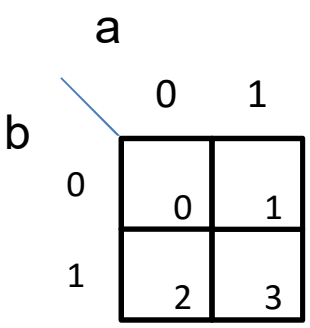
$S=\prod(0,3,5,6)$ – gdzie liczby oznaczają numer kolejny sumy (maxterm) dla której wartość funkcji =0

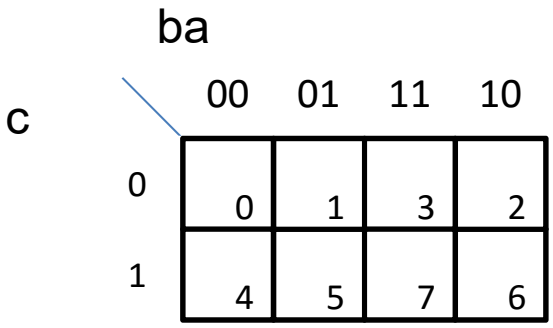
Minimalizacja wyrażeń logicznych

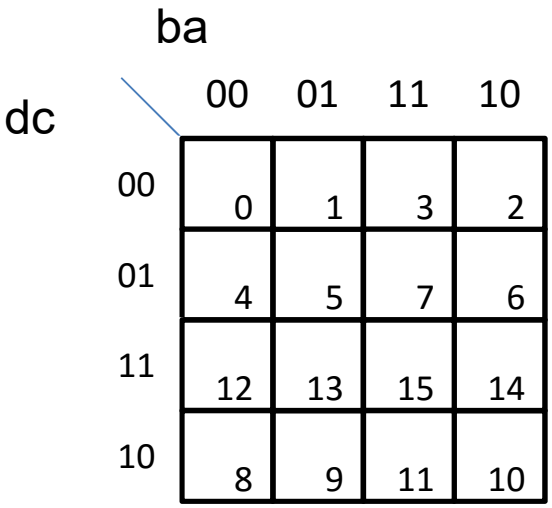
- Postać kanoniczna nie jest najprostsza
- Kryterium kosztu:
  - Redukcja liczby składników funkcji (liczba bramek)
  - Redukcja liczby literałów (liczba wejść bramek)
- Przekształcanie postaci kanonicznej do postaci równoważnej – tańszej wg przyjętej funkcji kosztu
- Przykład:
  - f(a,b,c,d)= $\cup$(5,7,13,15)= d'cb'a+d'cba+dcb'a+dcba=ca
  - Minimalizacja liczby składników z 4 do 1 i liczby literałów z 4 do 2
  - Zapis funkcji f()= $\cup$(5,7,13,15)+d(1,3,4) oznacza brak konkretnego wymagania na wartość funkcji (dowolna wartość 0 lub 1) dla kombinacji wejść 1,3 i 4.

Siatka Karnaugha

- Założenia:
  - waga zmiennych ustalona np. : od najniższej wagi a,b,c,d
- Dla n zmiennych: Prostokątna tablica zawierająca 2^n pól, każde pole reprezentuje jeden minterm (maxterm), mintermy odpowiadające sąsiednim polom różnią się wartością tylko jednej zmiennej.

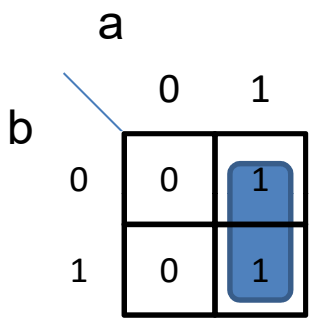




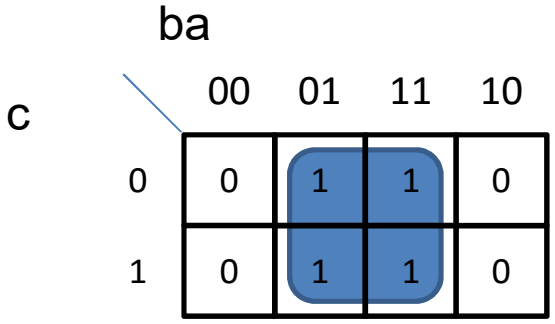


Twierdzenie o minimalizacji – reguła sklejania

- ab+ab’=a(b+b’)=a



f(a,b)=)= ∪(1,3)= ab’+ab=a

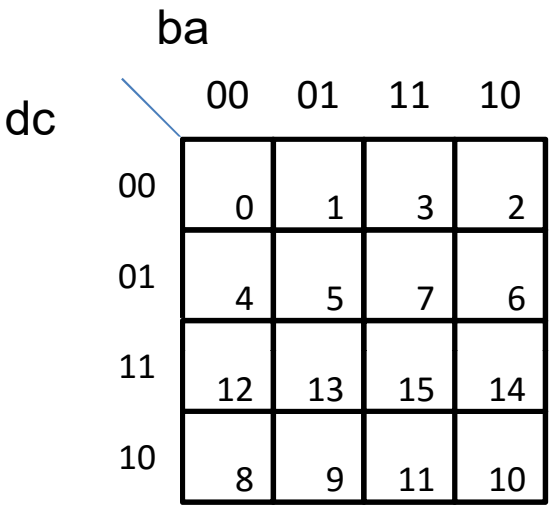


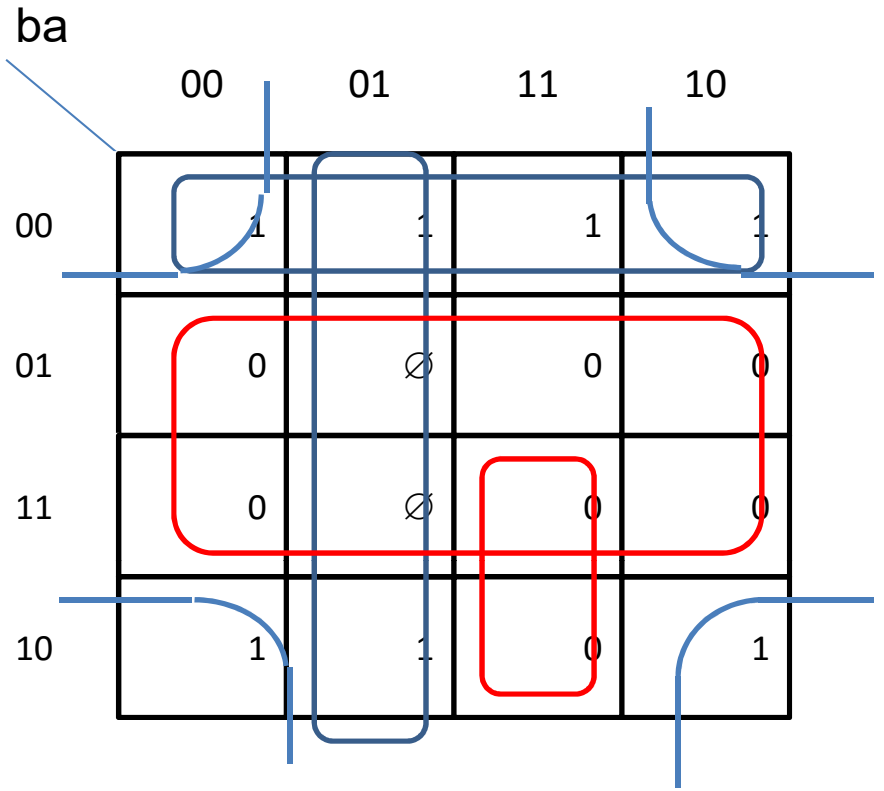
F(a,b,c)=)=
∪(1,3,5,7)=c’b’a+c’ba+cb’a+cba=
c’a+ca=a

Metoda tablic Karnaugha minimalizacji funkcji logicznej

- TABLICE. Przygotowanie tablic dla danej liczby zmiennych i wpisanie wartości w polach. W polach w krtórych wartość jest nieokreslona należy wpisać symbol nieokreśloności np. ∅
- SKLEJENIA. Narysować obwiednie łączące pola tworzące możliwie największe obszary. Obwiednie łączą sąsiednie pola z jedynkami (dla postaci sumacyjnej funkcji) [pola z zerami (dla postaci iloczynowej funkcji)] . Sąsiedztwo także cykliczne. Obwiednie pokrywają grupy pól tworzące prostokąt.
- Funkcja. Zapisanie postaci minimalnej funkcji w oparciu o wykonane sklejenia (obwiednie), każdy minterm (maxterm) musi być pokryty przez grupę uwzględnioną w zapisie. Tablica pokrycia.
- Uwaga: Pola ze znakami nieokreśloności można łączyć z dowolnymi innymi polami (jedynek lub zer w zależności od postaci funkcji) dla uzyskania maksymalnych sklejeń.

Minimalizacja – sklejenia dla jedynek i zer, funkcja dopełnieniowa





Funkcja f(a,b,c,d)= ∪(0,1,2,3,8,9,10)+d(5,13)

f=c'd' +c'a'+b'a grupa pozioma, grupa narożna, grupa pionowa

f=c' (d'+b'+a')

Terminologia minimalizacji

- Implikant:
  - każdy minterm lub
  - grupa mintermów które można połączyć.

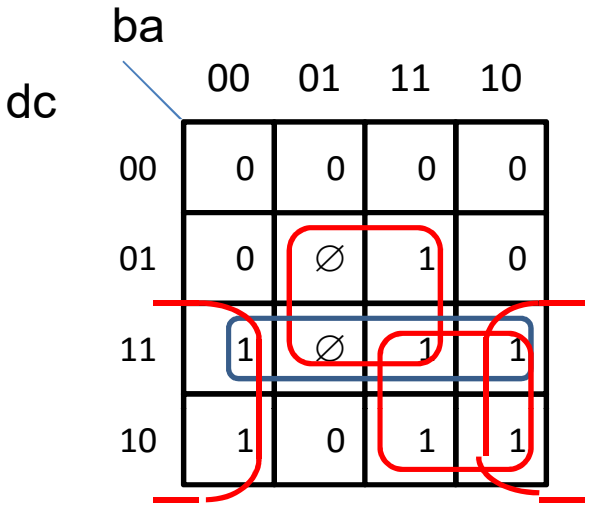
- Implikant prosty: implikant, któego nie można rozszerzyć przez sklejenia w tablicy Karnaugha.
- Implikant istotny: implikant prosty zawierający minterm nie występujący w żadnym implikancie prostym.

Metoda minimalizacji dwupoziomowej

1. Wygeneruj wszystkie implikanty proste
2. Utwórz pokrycie funkcji (mintermów) za pomocą minimalnej liczby implikantów.

Uwaga: Implikanty istotne są koniecznymi elementami pokrycia funkcji

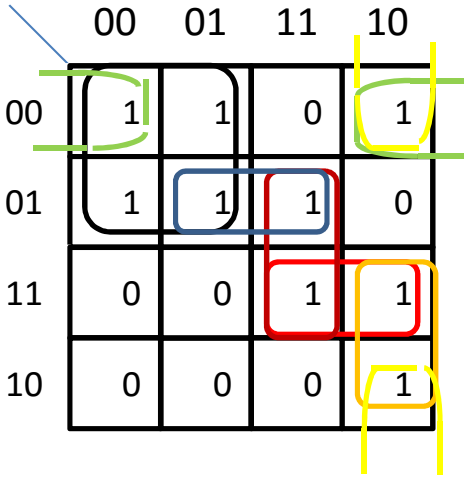
Przykład 1



- Implikanty proste: ca, dc, db, da’
- Implikanty istotne: ca, da’,db
- Implikanty istotne wystarczą do minimalnego pokrycia funkcji
- F(d,c,b,a)= ca+da’+db
- F(d,c,b,a)= (d+c)(a+d)(a’+b)

Przykład 1

- Realizacja funkcji na bramkach NAND bądź NOR
- F(d,c,b,a)= (ca+da’+db)’’=((ca)’(da’)’(db)’)’
- F(d,c,b,a)= (d+c)(a+d)(a’+b)’’= ((d+c)’+(a+d)’+(a’+b)’)’



Przykład 2 metoda Petricka

- Wyznaczenie minimalnego zbioru implikantów prostych – metoda Petricka
- Przykład:
  - Jeden implikant istotny
  - 5 implikantów prostych można wykorzystać do pokrycia 5 mintermów
  - Pokrycie wystąpi, gdy zastosujemy implikanty dla których funkcja Petricka przyjmuje wartość jeden
  - $P_x = 1$ gdy implikant x jest używany, 0 gdy imlikant x nie jest używany
  - $FP=(P_0+P_1)(P_1+P_2)(P_2+P_3)(P_3+P_4)(P_4+P_5) = P_1P_3P_5+P_1P_2P_4+P_0P_2P_4$

Metoda Quine’a-McCluskeya

1. Utwórz grupy mintermów odpowiadające liczbie jedynek w ich reprezentacji binarnych. Utworzenie początkowych implikantów.
2. Utwórz wszystkie implikanty przez połączenie implikantów jednej grupy z implikantami kolejnej grupy – jest to możliwe jeżeli różnią się wartością jednej zmiennej, zaznacz wykorzystane do łaczenia implikanty.
3. Powtarzaj krok 2 bazując na implikantach uzyskanych w poprzedniej iteracji 2 kroku.
4. Niewykorzystane w połączeniach implikanty tworzą zbiór implikantów prostych. Wybierz minimalny zbiór implikantów prostych.

Metoda Quine'a-McCluskeya

generacja implikantów prostych

wygodna dla funkcji wielu zmiennych

Funkcja f(a,b,c,d)= $\cup$(0,1,2,3,8,9,10)+d(5,13)

1	2	3
0 0000 √	0,1 000- √	0,1,2,3 00—
	0,2 00-0 √	0,2,8,10 -0-0
1 0001 √	0,8 -000 √	0,1,8,9 -00-
2 0010 √		
8 1000 √	1,3 00-1 √	1,5,9,13 –01
	1,5 0-01 √	
3 0011 √	1,9 -001 √	Implikanty proste
5 0101 √	2,3 001- √	00– d'c'
9 1001 √	2,10 -010 √	-0-0 c'a'
10 1010 √	8,9 100- √	-00- c'b'
	8,10 10-0 √	--01 b'a
13 1101 √		
	5,13 -101 √	
	9,13 1-01 √	

Metoda Quine'a-McCluskeya tablica pokrycia mintermów

	0	1	2	3	8	9	10
0,1,2,3 00—	√	√	√	√			
0,2,8,10 -0-0	√		√		√		√
0,1,8,9 -00-	√	√			√	√	
1,5,9,13 –01		√				√	

W kolumnach tablicy uwzględniamy tylko mintermy z określonymi dla funkcji wartościami

Implikanty istotne

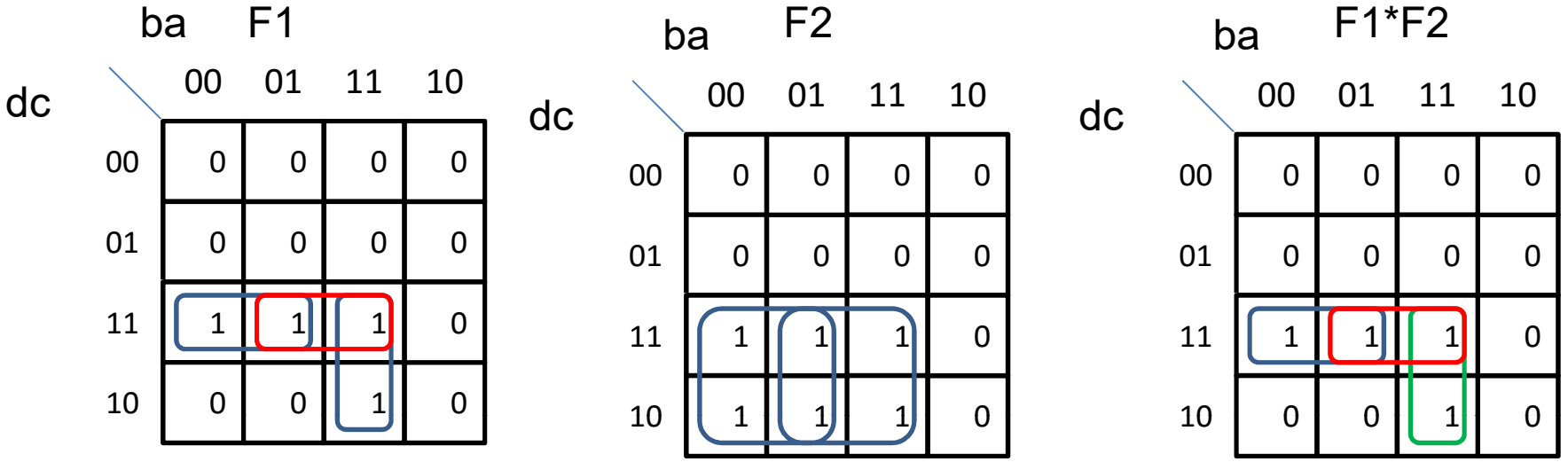
Mintermy pokryte przez implikanty istotne

Możliwe warianty funkcji o minimalnej liczbie implikantów:

F=d'c'+c'a' + c'b'

F=d'c'+c'a' + b'a

Minimalizacja funkcji wielowyjściowych



- Wyznaczenie implikantów prostych dla: funkcji optymalizowanych i wszystkich iloczynów funkcji - (powyżej 6 implikantów prostych w 3 grupach).
- Znajdowanie pokrycia minimalną liczbą spośród wszystkich implikantów (tablica pokrycia): implikant iloczynu dwóch funkcji (zielony) pokrywa mintermy obu funkcji

Komputerowo wspomaganie minimalizacji funkcji logicznych

- Znalezienie pokrycia minimalnego jest problemem NP-trudnym.
- Ze względu na trudność problemu dla dużych instancji stosowane są metody przybliżone.
  - brak generacji wszystkich implikantów
  - zapewnienie pokrycia funkcji przez wybrany zbiór implikantów