

# Podstawy techniki cyfrowej

## zima 2018

Wykład

dr inż. Rafał Walkowiak

# Literatura

1. Podstawy Techniki Cyfrowej, Barry Wilkinson, WKiŁ 2001
2. Podstawy projektowania układów logicznych i komputerów, M.M.Mano, Ch.R.Kime, WNT 2007
3. Komputerowe projektowanie układów cyfrowych, T.Łuba, B.Zbierzchowski, WKiŁ, 2000
4. Podstawy projektowania układów cyfrowych, Cezary Zieliński, PWN 2012
5. Język VHDL: projektowanie programowalnych układów logicznych, Kevin Shakill, WNT 2004
6. Układy cyfrowe, Zbiór zadań z rozwiązaniami, J.Tyszer, G.Mrugalski, Wydawnictwo PP
7. Układy Scalone TTL w systemach cyfrowych, J. Pienkos, J. Turczyński, WkiŁ, 1994

# Zakres przedmiotu

- Wstęp: arytmetyka binarna, algebra Boole'a , kody binarne, BCD, podstawowe funkcje logiczne, sposoby przedstawiania funkcji logicznych - postaci 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: zatrask RS, zatrask D, przerzutniki: D, JK, T; parametry czasowe, rejestry szeregowo, równoległe, przesuwne, rejestry liczące.
- Liczniki: synchroniczne i asynchroniczne, binarne, dziesiętne; łączenie liczników, synteza liczników, skracanie liczników, taktowanie systemów cyfrowych, częstotliwości maksymalne liczników;
- Automaty synchroniczne: Moore, 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.

# Podstawowe informacje organizacyjne

- Zaliczenie: 2 sprawdziany na ćwiczeniach i egzamin
- Ocena - średnia ważona (2 składniki – ćwiczenia, 3 składniki – egzamin).
- Sprawdzian z określonego „materiału ćwiczeń” : na ćwiczeniach i na ostatnim wykładzie.
- Egzamin w sesji.
- Warunki konieczne do zwolnienia z egzaminu : bardzo dobre zaliczenie ćwiczeń i min 80 % obecności na wykładach ze sprawdzaną obecnością.

# 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. liczb w różnych systemach liczbowych).

# Systemy liczbowe

Pozycyjne systemy liczbowe – np. dziesiętny, dwójkowy (czyli binarny), ósemkowy, szesnastkowy – systemy o podstawie 10, 2, 8 lub 16 zawierają zbiór cyfr o liczności równej tzw. podstawie systemu.

NKB

- System pozycyjny o podstawie systemu równej 2
- Liczby określone są bez znaku np. 1010  $b_1 = b_3 = 1$   $b_2 = b_0 = 0$
- Wartość dziesiętna liczby binarnej (N- długość słowa kodowego,  $b_i$  – cyfra pozycji  $i$ ) =  $\sum_{i=0, n-1} 2^i \cdot b_i$
- Wartość cyfry  $b_i$  zależy od jej pozycji  $i$  w liczbie
- wartość  $b_i = 2^i$  ( $i$  numerowanie od zera)
- $2^N$  różnych wartości kodu (kod pełny)

Konwersje liczb między systemami binarnym i dziesiętnym, konwersje liczb ułamkowych.

# Systemy liczbowe uzupełnieniowe

- **System liczbowy „Uzupełnienie do K”** - Uzupełnienie do K liczby **N** o **n** cyfrach zapisanej w systemie o podstawie K definiujemy jako liczbę równą:

$$K^n - N$$

liczba w kodzie **Uzupełnienie do K** jest reprezentowana przez wartość różnicy  $K^n$  i  $N$  – wartość ta jest wyrażona w systemie o podstawie K (czyli za pomocą K cyfr)

Np. liczba dziesiętna 345 to trzy cyfrowa  $K=10$   $n=3$

liczba z systemu dziesiętnego 345 zapisana w systemie uzupełnieniowym do 10 ma postać 655,

liczba 345 w systemie dziesiętnym:  $345=000345$  (zera nieznaczące)

Liczba 345 w systemie uzupełnienie do 10:  $655=99655$  (dziewiątki nieznaczące)

Liczba dwójkowa 101 w systemie uzupełnienie do 2 ( $U_2$ ) ma postać 011 lub 111011 (jedyńki nieznaczące) wartość 3 uzupełnia 5 do 8, a wartość 59 uzupełnia 5 do 64

# Reprezentacja uzupełnieniowa

- Binarna liczba **dodatnia** jest zapisywana w NKB na wystarczającej liczbie pozycji i uzupełniana **zerami** na pozycjach bardziej znaczących:  $(3)_{10} = (011)_2 = (0011)_2$  – najstarsze zero jest bitem znaku
- Binarna liczba **ujemna** jest zapisywana:
  - w kodzie **uzupełnienie do 2** i
  - poprzedzona 1 (dodatkowo jeśli potrzeba) na pozycji znaku i
  - uzupełniona **jedynkami** na pozycjach bardziej znaczących:  $(-3)_{10} = (101)_{UZ} = (1101)_{UZ}$
  - Gdy najstarszy bit kodu  $U2 = 1$  to staje się bitem znaku i nie potrzeba umieszczać 1 przed kodem  $U2$  dla  $-8_D = 1000_{UZ}$
- **Notacja uzupełnieniowa liczb binarnych pozwala na dodawanie liczb dodatnich i ujemnych realizowane przez sumator zaprojektowany dla liczb wyrażonych w NKB.**

# Reprezentacje liczb binarnych ze znakiem

- Reprezentacja liczby binarnej **znak moduł** (ZM) – najstarszy bit określa znak liczby, pozostałe bity w NKB, bit znaku 0 (liczba dodatnia) lub 1 (liczba ujemna)
- **Reprezentacja uzupełnieniowa** liczby binarnej (RU) korzysta z U2
  - bit znaku (0) i moduł liczby dodatniej w NKB,
  - bit znaku (1) i moduł liczby ujemnej w kodzie U2 (najbardziej znaczące jedyńki można usunąć z zapisu)
  - Przykład  $-8$ , moduł 8  $u_2(8)=1000$   $-8=RU(11000)=RU(1000)$   $8=RU(01000)$

| N1 | ZM(N1) | RU(N1) |  | N2 | ZM(N2)=RU(N2) |
|----|--------|--------|--|----|---------------|
| -8 | -      | 1000   |  | 7  | 0111          |
| -7 | 1111   | 1001   |  | 6  | 0110          |
| -6 | 1110   | 1010   |  | 5  | 0101          |
| -5 | 1101   | 1011   |  | 4  | 0100          |
| -4 | 1100   | 1100   |  | 3  | 0011          |
| -3 | 1011   | 1101   |  | 2  | 0010          |
| -2 | 1010   | 1110   |  | 1  | 0001          |
| -1 | 1001   | 1111   |  | 0  | 0000          |

# Dodawanie liczb ujemnych wykorzystanie notacji UZ

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

|      |        |
|------|--------|
| 1101 | -3     |
| 0010 | $+2$   |
| 1111 | $= -1$ |

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

Przeniesienie jest ignorowane, wynik poprawny gdy dwa przeniesienia: na najstarszy bit i z najstarszego bitu są jednakowe.

# Odejmowanie liczb – dodawanie liczby przeciwnej

$$\begin{array}{r} 0011 \text{ (3d)} \\ + 1011 \text{ (-5d)} \\ \hline \end{array}$$

$$1110 \text{ (-2d)}$$

$$\begin{array}{r} 0101 \text{ (5d)} \\ + 1101 \text{ (-3d)} \\ \hline \end{array}$$

$$0010 \text{ (2d)}$$

Binarna liczba ujemna – bit znaku i liczba binarna w uzupełnieniu do 2

$$00010110 = 22 \text{ (d)}$$

Wyznaczenie liczby w kodzie U2 –

Metoda 1:

11101001 negacja bitów

**111010**10 dodanie jedynki = -22 (d)

Metoda 2:

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

100 -> 100 dodatkowy bit niepotrzebny dla -4

101 -> 011 konieczny dodatkowy bit dla -5 -> 1011

# 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

$$\begin{array}{r} 0011 \text{ (3d)} \\ + 0011 \text{ (3d)} \\ \hline \end{array}$$

0110 (6d) wynik dodatni –  
poprawnie

$$\begin{array}{r} 0101 \text{ (5d)} \\ + 0101 \text{ (5d)} \\ \hline \end{array}$$

1010 -(6d) Wynik ujemny  
- **niepoprawny**

$$\begin{array}{r} 1101 \text{ (-3)} \\ + 1101 \text{ (-3)} \\ \hline \end{array}$$

1010 (-6) wynik ujemny –  
poprawnie

$$\begin{array}{r} 1011 \text{ (-5)} \\ + 1011 \text{ (-5)} \\ \hline \end{array}$$

0110 (6) wynik dodatni -  
**niepoprawny**

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

# Kody dwójkowe niewagowe

pozycja binarna nie posiada wagi

| Kod<br>cyfra | Z<br>nadmiarem<br>3 | Graya | Wattsa | Johnsona | Wskaźników<br>7<br>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 dwójkowo-dziesiętne

- Do reprezentacji cyfr dziesiętnych
- 10 cyfr dziesiętnych (0,1,2,3,4,5,6,7,8,9) zakodowanych za pomocą ciągu 4 bitów (ciąg ten dostarcza 16 kombinacji na 4 bitach) – 6 kombinacji jest niewykorzystanych.

Warianty kodów dwójkowo-dziesiętnych:

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

# Kody dwójkowo-dziesiętne wagowe

| kod   | Naturalny<br>NKB |       | Aikena |      |        |
|-------|------------------|-------|--------|------|--------|
| Wagi  | 8421             | 2*421 | 2421   | 7421 | 84-2-1 |
| cyfra |                  |       |        |      |        |
| 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 detekcyjne

| kod          | 1 z 10     | 2 z 5     | 2 z 7   | Bin z Bitem parzystości |
|--------------|------------|-----------|---------|-------------------------|
| Wagi-> cyfra | 9876543210 | niewagowy | 5043210 | 8421 BP                 |
| 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.

# Cyfry dziesiętne kodowane w NKB – kod BCD 8421

- Dziesiętny charakter informacji lecz kodowanie cyfr w NKB
- 2345 (10) = 0010 0011 0100 0101 (BCD) 4 pozycje (4 bitowe) cyfr dziesiętnych
- Dodawanie liczb w kodzie BCD realizowane tak jak dodawanie liczb binarnych, **lecz**:
  - wystąpienie podczas dodawania liczb przeniesienia na pozycję kolejnej cyfry dziesiętnej (kolejne 4 bity) wymaga skorygowania (czyli dodania wartości 6) na tej pozycji, z której przeniesienie wystąpiło
  - wystąpienie wyniku na 4 bitach (jednej pozycji cyfry dziesiętnej) spoza zakresu (10-15) wymaga skorygowania wyniku czyli dodania wartości 6 na tej pozycji cyfry dziesiętnej, która nie jest poprawna; może wystąpić przeniesienie, które należy uwzględnić na kolejnej pozycji (ale bez korekcji pozycji bieżącej), możliwa również propagacja przeniesienia np. dla liczb 3456 + 6545.

# Dodawanie w kodzie BCD

|       |   |             |                                         |
|-------|---|-------------|-----------------------------------------|
| 89    |   | 1000 1001   |                                         |
| +18   | + | 0001 1000   |                                         |
| ----- |   |             |                                         |
| 107   |   | 1010 0001   | przeniesienie -> korekcja               |
|       |   | 0110        |                                         |
| ----- |   |             |                                         |
|       |   | 1010 0111   | wartość spoza przedziału ->korekcja     |
|       |   | 0110        |                                         |
| ----- |   |             |                                         |
|       |   | 1 0000 0111 | przeniesienie bez konieczności korekcji |

# 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 <sub>7</sub>                                              | 0   | 0    | 0                   | 0               | 1               | 1               | 1               | 1               |
| b <sub>6</sub>                                              | 0   | 0    | 1                   | 1               | 0               | 0               | 1               | 1               |
| b <sub>5</sub>                                              | 0   | 1    | 0                   | 1               | 0               | 1               | 0               | 1               |
| b <sub>4</sub> b <sub>3</sub> b <sub>2</sub> b <sub>1</sub> |     |      |                     |                 |                 |                 |                 |                 |
| 0 0 0 0                                                     | NUL | DLE  | SPACE <sup>2)</sup> | 0 <sup>2)</sup> | @ <sup>1)</sup> | P               | \ <sup>1)</sup> | 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 <sup>1)</sup>     | 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               | [ <sup>1)</sup> | k               | { <sup>1)</sup> |
| 1 1 0 0                                                     | FF  | FS   | „                   | <               | L               | \ <sup>1)</sup> | l               | <sup>1)</sup>   |
| 1 1 0 1                                                     | CR  | GS   | —                   | =               | M               | ]               | m               | }               |
| 1 1 1 0                                                     | SO  | RS   | .                   | >               | N               | ^ <sup>1)</sup> | n               | _ <sup>1)</sup> |
| 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  
 ENQ — 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) – „ $\cdot$ ” „ $\cap$ ” „ $\wedge$ ” (alternatywne oznaczenia)
- Suma logiczna LUB (OR) – „ $+$ ” „ $\cup$ ” „ $\vee$ ” (alternatywne oznaczenia)
- Negacja NIE (NOT) – „**linia nad zmienną**” „ $'$ ” (alternatywne oznaczenia)

Funkcja boolowska (logiczna, przełączająca) – 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**:  $a + b \in Z$   $A \cdot B \in Z$

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:  $+\rightarrow \cdot$  ,  $\cdot \rightarrow +$  ,  
 $0 \rightarrow 1$  ,  $1 \rightarrow 0$

Wartościowania (prawda, fałsz) wyrażenia prostego i dualnego jest jednakowe.

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 – tańsza implementacja ma mniej składników oraz/lub składniki prostsze.

# Twierdzenia algebry Boole'a

- **Idempotentność** (łac. taki sam) –

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

- **Jednoznaczność negacji** –

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

- **Dominacja** - dla każdego  $a$      $a \cdot 0 = 0$      $a+1=1$

- **Podwójna negacja** –

dla każdego  $a$  zachodzi     $a = \overline{\bar{a}}$

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

# Twierdzenia algebry Boole'a

- Uproszczenie

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

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

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

- Łączność -  $(a+b)+c=a+(b+c)$      $(a \cdot b) \cdot c = a \cdot (b \cdot c)$

- Konsensus (zgoda) -

Wystarczy jedna 1 dla b i c,     $a \cdot b + \bar{a} \cdot c + b \cdot c = a \cdot b + \bar{a} \cdot c$

wystarczy jedno 0 dla b i c

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

# Prawo de Morgana

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

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

# Funkcje logiczne dwóch zmiennych i ich wartości

zmienne binarne a b

| Wartości            | ab | ab | ab | ab | Równanie                      | Nazwa                        | Skrót |
|---------------------|----|----|----|----|-------------------------------|------------------------------|-------|
| argumentów          | 00 | 01 | 10 | 11 | funkcji                       | funkcji                      | Nazwy |
| Wartości<br>funkcji | 0  | 0  | 0  | 0  | 0                             | Stała Zero                   | AND   |
|                     | 0  | 0  | 0  | 1  | $a \cdot b$                   | Iloczyn logiczny             |       |
|                     | 0  | 0  | 1  | 0  | $a \cdot b'$                  | Zakaz przez b                |       |
|                     | 0  | 0  | 1  | 1  | a                             | Identyczna z a               |       |
|                     | 0  | 1  | 0  | 0  | $a' \cdot b$                  | Zakaz przez a                |       |
|                     | 0  | 1  | 0  | 1  | b                             | Identyczna z b               |       |
|                     | 0  | 1  | 1  | 0  | $(a' \cdot b) + (a \cdot b')$ | 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  | $(a \cdot b) + (a' \cdot b')$ | Równoważność                 | EQU   |
|                     | 1  | 0  | 1  | 0  | $b'$                          | Negacja b                    | NAND  |
|                     | 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  | $(a \cdot b)'$                | negacja iloczynu             |       |
|                     | 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 (suma wyłączna) i XNOR (równoważność):
  - $a \oplus b = a'b + b'a = (a+b)(a'+b')$
  - $(a \oplus b)' = a' \oplus b = b' \oplus a = ab + a'b' = (a'+b)(a+b')$
  - $a \oplus 1 = a'$      $a \oplus 0 = a$
- Różne interpretacje logiczne wielowejsciowych 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

np.

| Nr kombinacji | $x_0 x_1 x_2 \dots x_{n-1}$ | f                   |
|---------------|-----------------------------|---------------------|
| 0             | 0 0 0 *** 0                 | Wartości<br>funkcji |
| 1             | 0 0 0 *** 1                 |                     |
| 2             |                             |                     |
| 3             |                             |                     |
| 4             |                             |                     |
| 5             |                             |                     |
| *             |                             |                     |
| *             |                             |                     |
| *             |                             |                     |
| $2^n - 1$     | 1 1 1 *** 1                 |                     |

| 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 Karnaugh
- Kombinacji wejść odpowiada **pole tablicy**, w polu umieszczamy właściwą dla kombinacji wartość.
- Sąsiednie (w poziomie i pionie – także cyklicznie) pola tablicy Karnaugh odpowiadają kombinacji argumentów różniące 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 Karnaugh

a

b

|   | 0 | 1 |
|---|---|---|
| 0 | 0 | 1 |
| 1 | 0 | 1 |

c

ba

|   | 00 | 01 | 11 | 10 |
|---|----|----|----|----|
| 0 | 0  | 1  | 1  | 0  |
| 1 | 0  | 1  | 1  | 0  |

ba

dc

|    | 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:

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

- Gdzie:  $\bigvee$  to suma
- $I_j$  oznacza minterm - **iloczyn** zmiennych niezależnych dla j-tej kombinacji zmiennych =1; zawiera zmienną prostą gdy bit zmiennej jest równy 1 lub zmienną zanegowaną gdy bit zmiennej jest równy 0
  - np. zerowa kombinacja wejść: 0000: minterm -  $x_0'x_1'x_2'x_3'$   
(wartość wyrażenia dla tej kombinacji wartości zmiennych wynosi jeden)
- $a_j$  wartość funkcji odpowiadająca j-tej kombinacji zmiennych
- term – wyrażenie składające się ze zmiennych i symboli funkcyjnych

# Dysjunkcyjna postać kanoniczna – przykład

| Iloczyny zmiennych | $abC_{in}$ | 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 (mintermu) dla którego wartość funkcji =1

(należy określić wagę (1,2,4) zmiennej; w przykładzie: **a** jest najbardziej znaczącym bitem - waga =4)

# Sposoby przedstawiania funkcji logicznych

- Koniunkcyjna (iloczynowa) postać kanoniczna:

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

- Gdzie:
- $S_j$  oznacza maxterm - **sumę** zmiennych niezależnych dla j-tej kombinacji zmiennych = 0; zawiera zmienną prostą gdy bit tej zmiennej jest równy 0 lub zmienną zanegowaną gdy bit kombinacji jest równy 1
  - Np. zerowa kombinacja wejść : 0000; suma dla tej kombinacji:  $x_0+x_1+x_2+x_3$ , wartość wyrażenia dla tej kombinacji zmiennych wynosi 0
- $a_j$  oznacza wartość funkcji odpowiadającej j-tej kombinacji zmiennych.

# Konjunktcyjna postać kanoniczna - przykład

| sumy                      | abC <sub>in</sub> | S | Cout |
|---------------------------|-------------------|---|------|
| 0 a+b+c <sub>in</sub>     | 000               | 0 | 0    |
| 1 a+b+c <sub>in</sub> '   | 001               | 1 | 0    |
| 2 a+b'+c <sub>in</sub>    | 010               | 1 | 0    |
| 3 a+b'+c <sub>in</sub> '  | 011               | 0 | 1    |
| 4 a'+b+c <sub>in</sub>    | 100               | 1 | 0    |
| 5 a'+b+c <sub>in</sub> '  | 101               | 0 | 1    |
| 6 a'+b'+c <sub>in</sub>   | 110               | 0 | 1    |
| 7 a'+b'+c <sub>in</sub> ' | 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)
- Minimalizacja to 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 1,3 i 4 kombinacji wejść.

# Siatka Karnaugh

- 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.

a

b

|   |   |   |
|---|---|---|
|   | 0 | 1 |
| 0 | 0 | 1 |
| 1 | 2 | 3 |

c

ba

|   |    |    |    |    |
|---|----|----|----|----|
|   | 00 | 01 | 11 | 10 |
| 0 | 0  | 1  | 3  | 2  |
| 1 | 4  | 5  | 7  | 6  |

dc

ba

|    |    |    |    |    |
|----|----|----|----|----|
|    | 00 | 01 | 11 | 10 |
| 00 | 0  | 1  | 3  | 2  |
| 01 | 4  | 5  | 7  | 6  |
| 11 | 12 | 13 | 15 | 14 |
| 10 | 8  | 9  | 11 | 10 |

# Twierdzenie o minimalizacji – reguła sklejania

- $ab+ab'=a(b+b')=a$

a

b

|   |   |   |
|---|---|---|
|   | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 1 | 0 |

$$f(a,b) = \Sigma(0,2) = a'b' + a'b = a'(b+b') = a'$$

Uzasadnienie do sklejania sąsiednich pól siatki.

Powstającą grupę opisuje wyrażenie iloczynowe posiadające mniejszą liczbę zmiennych (usuwamy z opisu grupy tę zmienną, która dla pól przyjmuje różne wartości – b, a pozostaje a), zmienna, która dla grupy przyjmuje wartość 0 pozostaje w opisie (iloczynowym) grupy jako zmienna zanegowana ( 1 – zmienna prosta)

ba

c

|   |    |    |    |    |
|---|----|----|----|----|
|   | 00 | 01 | 11 | 10 |
| 0 | 0  | 1  | 1  | 0  |
| 1 | 0  | 1  | 1  | 0  |

$$F(a,b,c) = \Sigma(1,3,5,7) = c'b'a + c'ba + cb'a + cba = c'a + ca = a$$

Sklejamy poziomo usuwając zmienną b, sklejamy pionowo usuwając zmienną c, pozostaje tylko a dla opisu grupy, a- proste gdyż przyjmuje wartość 1 dla wszystkich pól.

# Twierdzenie o minimalizacji – reguła sklejania

- $F(a,b) = \prod(0,1) = (a'+b)(a+b) = a'a + a'b + ab + b = b$

Diagram Karnaugh dla funkcji  $F(a,b)$ . Oś pozioma to  $a$  (0, 1), oś pionowa to  $b$  (0, 1).

|   |   |   |   |
|---|---|---|---|
|   | a | 0 | 1 |
| b | 0 | 0 | 0 |
| 1 | 1 | 1 | 1 |

Wartości 0 dla  $b=0$  są zaznaczone w niebieskim prostokącie, co oznacza grupę  $b=0$ .

Uzasadnienie do sklejania sąsiednich pól siatki. Powstającą grupę opisuje wyrażenie sumacyjne posiadające mniejszą liczbę zmiennych (usuwamy z opisu grupy tę zmienną, która dla sąsiednich pól przyjmuje różne wartości -  $a$ , a pozostaje  $b=0$ ), zmienna, która dla grupy przyjmuje wartość 0 pozostaje w opisie (sumacyjnym) grupy jako zmienna prosta (1 – zmienna zanegowana)

Diagram Karnaugh dla funkcji  $F(a,b,c)$ . Oś pozioma to  $ba$  (00, 01, 11, 10), oś pionowa to  $c$  (0, 1).

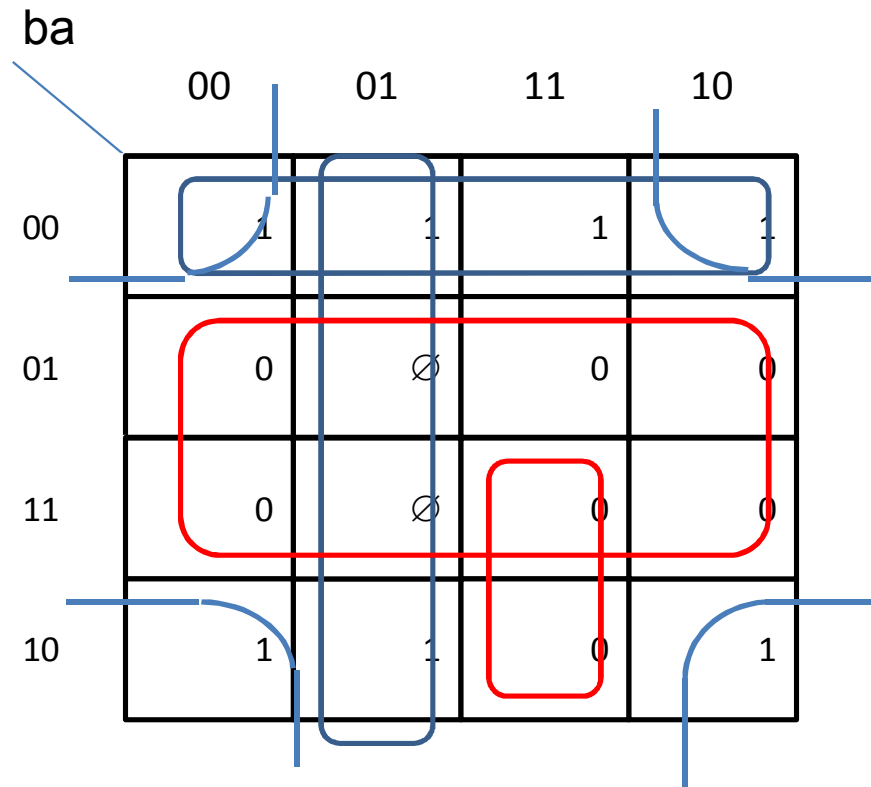
|   |    |    |    |    |    |
|---|----|----|----|----|----|
|   | ba | 00 | 01 | 11 | 10 |
| c | 0  | 0  | 1  | 1  | 0  |
| 1 | 1  | 0  | 1  | 1  | 0  |

Wartości 0 dla  $c=0$  i  $c=1$  są zaznaczone w niebieskich prostokątach, co oznacza grupę  $c=0$ . Niebieska łukowata strzałka pod tabelą wskazuje na te grupy.

$F(a,b,c) = \prod(0,2,4,6) = (a+b+c)(a+b'+c)(a+b+c')(a+b'+c')$   
 $= a$  Sklejamy poziomo usuwając zmienną  $b$ , sklejamy pionowo usuwając zmienną  $c$ , dla opisu grupy pozostaje tylko  $a$ , proste gdyż przyjmuje wartość 0 (dla wszystkich pól).

# Minimalizacja – sklejenia dla jedynek i zer, funkcja dopełnieniowa

|    |    |    |    |    |    |
|----|----|----|----|----|----|
|    |    | ba |    |    |    |
|    |    | 00 | 01 | 11 | 10 |
| dc | 00 | 0  | 1  | 3  | 2  |
|    | 01 | 4  | 5  | 7  | 6  |
|    | 11 | 12 | 13 | 15 | 14 |
|    | 10 | 8  | 9  | 11 | 10 |



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

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

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

# Metoda tablic Karnaugh minimalizacji funkcji logicznej

- TABLICE. Przygotowanie tablic dla danej liczby zmiennych i wpisanie wartości w polach. W polach w których wartość jest nieokreślona należy wpisać symbol nieokreśloności np.  $\emptyset$
- SKLEJENIA. Narysować obwiednie łączące pola tworzące możliwie największe obszary. **Obwiednie łączą sąsiednie pola z jedynekami (dla postaci sumacyjnej funkcji)** [pola z zerami (dla postaci iloczynowej funkcji)] . Sąsiedztwo także cykliczne. Obwiednie pokrywają grupy pól tworzące prostokąt (1,2,4,8,16... pól).
- Funkcja. Zapisanie postaci minimalnej funkcji w oparciu o wykonane sklejenia (obwiednie), (suma iloczynów- każda grupa wykorzystana do opisu to składnik sumy), każde pole z 1 musi być pokryte przez dowolną grupę uwzględnioną w zapisie.
- Uwaga: Pola ze znakami nieokreśloności można łączyć z dowolnymi innymi polami (jedynek lub zer w zależności od postaci funkcji) dla uzyskania maksymalnych sklejeń.

# Terminologia minimalizacji

- Implikant:
  - każdy minterm (iloczyn) dla którego funkcja jest równa 1 lub
  - grupa mintermów z wartością funkcji 1 lub  $\emptyset$  które można skleić.
- **Implikant prosty:** implikant, którego nie można rozszerzyć przez sklejenia w tablicy Karnaugh.
- **Implikant istotny:** implikant prosty zawierający ten minterm z wartością funkcji = 1, który nie występuje w żadnym innym implikancie prostym.

# Terminologia minimalizacji

- **Implicant:**
  - każdy maxterm (suma) dla którego funkcja jest równa 0 lub
  - grupa maxtermów z wartością funkcji 0 lub  $\emptyset$  które można skleić.
- **Implicant prosty:** Implicant, którego nie można rozszerzyć przez sklejenia w tablicy Karnaugh.
- **Implicant istotny:** Implicant prosty zawierający ten maxterm z 0, który nie występuje w żadnym innym implikancie prostym.

# Metoda minimalizacji dwupoziomowej (wersja suma iloczynów)

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

Tablica pokrycia. Dla określenia postaci funkcji **nie korzystającej** wyłącznie z implikantów kluczowych należy wykonać tablicę pokrycia.  
Uwaga: Implikanty istotne są koniecznymi elementami pokrycia funkcji.

# Metoda minimalizacji dwupoziomowej (wersja iloczyn sum)

1. Wygeneruj **wszystkie** implikenty proste.
2. Utwórz pokrycie funkcji (wartości z 0) za pomocą minimalnej liczby implikentów.

Tablica pokrycia. Dla określenia postaci funkcji **nie korzystającej** wyłącznie z implikentów kluczowych należy wykonać tablicę pokrycia.  
Uwaga: Implikenty istotne są koniecznymi elementami pokrycia funkcji

# Przykład 1

|    |    | ba |    |    |    |
|----|----|----|----|----|----|
|    |    | 00 | 01 | 11 | 10 |
| dc | 00 | 0  | 0  | 0  | 0  |
|    | 01 | 0  | ∅  | 1  | 0  |
|    | 11 | 1  | ∅  | 1  | 1  |
|    | 10 | 1  | 0  | 1  | 1  |

- Implikanty proste:  $ca$ ,  $dc$ ,  $db$ ,  $da'$
- Implikanty istotne:  $ca$ ,  $da'$ ,  $db$
- Implikanty istotne wystarczają 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
- przejście między rodzajami funkcji - zastosowanie prawa deMorgana

$$F(d,c,b,a) = (ca + da' + db)'' = ((ca)'(da')'(db)')'$$

$$F(d,c,b,a) = (d+c)(a+d)(a'+b)'' = ((d+c)' + (a+d)' + (a'+b)')'$$

# Metoda Quine'a-McCluskeya

1. Utwórz grupy kombinacji zmiennych (z wartością funkcji =1) posiadające jednakową liczbę 1 w ich reprezentacji binarnej. Jest to utworzenie początkowych implikantów.
2. Utwórz wszystkie implikanty przez połączenie implikantów jednej grupy z implikantami kolejnej grupy – jest to możliwe jeżeli reprezentacje binarne kombinacji zmiennych różnią się wartością jednej zmiennej, zaznacz wykorzystane do łączenia implikanty (nie będą już dalej wykorzystywane).
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 za pomocą tablicy pokrycia lub funkcji Petrica.

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

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

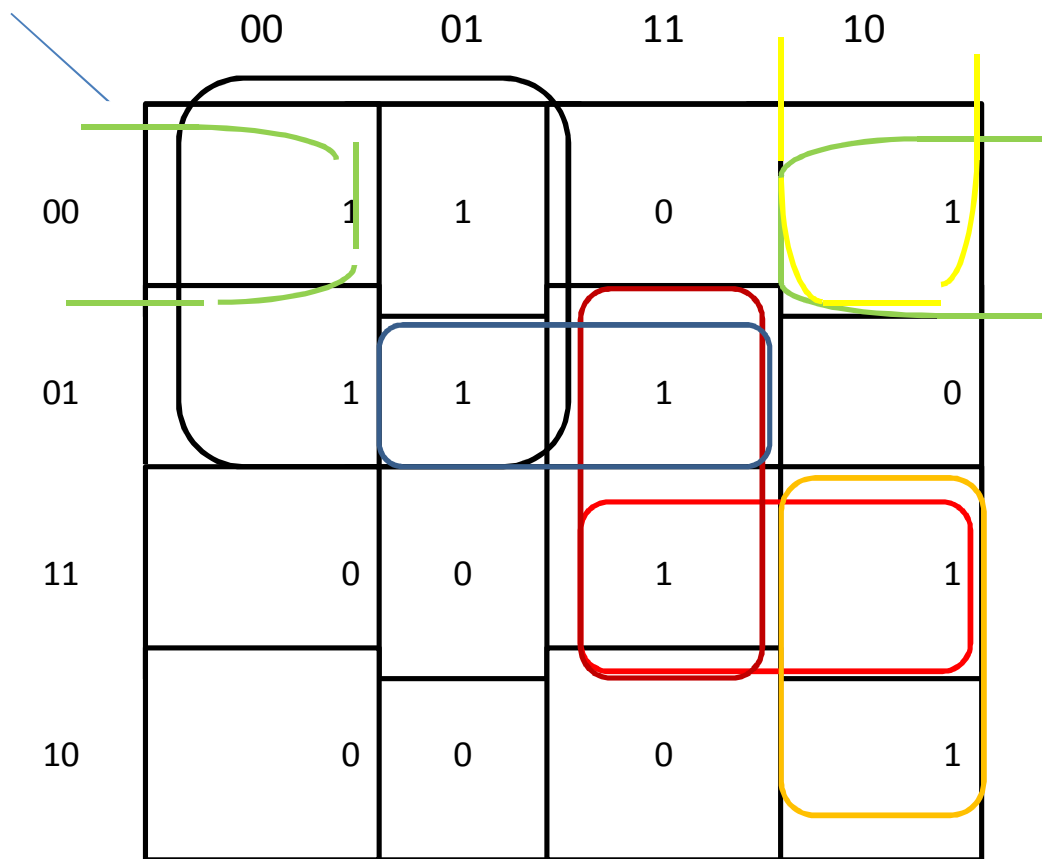
$$F = d'c' + c'a' + b'a$$

|    |    |    |    |    |
|----|----|----|----|----|
|    | 00 | 01 | 11 | 10 |
| 00 | 1  | 1  | 0  | 1  |
| 01 | 1  | 1  | 1  | 0  |
| 11 | 0  | 0  | 1  | 1  |
| 10 | 0  | 0  | 0  | 1  |

# Przykład 2 metoda Petricka

jeśli tablica pokrycia nie wystarcza

- Pozwala na wyznaczenie minimalnego zbioru implikantów prostych (nie dotyczy implikantów istotnych)
- 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
- $P_x$  w równaniu oznacza wykorzystanie implikanta  $x$  –
- $FP = (P_0 + P_1)(P_1 + P_2)(P_2 + P_3)(P_3 + P_4)(P_4 + P_5) = P_1P_3P_5 + P_1P_2P_4 + P_0P_2P_4$   
–
- Zapis czytamy : pokrycie wystąpi gdy użyjemy implikant (0 lub 1) i (1 lub 2) i (2 lub 3) i (3 lub 4) ... zapewnia to pokrycie odpowiednio 7,15,14,10 i 2 kombinacji z 1



# Minimalizacja funkcji wielowyjściowych

ba

F1

dc

00

01

11

10

|    |   |   |   |   |
|----|---|---|---|---|
| 00 | 0 | 0 | 0 | 0 |
| 01 | 0 | 0 | 0 | 0 |
| 11 | 1 | 1 | 1 | 0 |
| 10 | 0 | 0 | 1 | 0 |

ba

F2

dc

00

01

11

10

|    |   |   |   |   |
|----|---|---|---|---|
| 00 | 0 | 0 | 0 | 0 |
| 01 | 0 | 0 | 0 | 0 |
| 11 | 1 | 1 | 1 | 0 |
| 10 | 1 | 1 | 1 | 0 |

ba

F1\*F2

dc

00

01

11

10

|    |   |   |   |   |
|----|---|---|---|---|
| 00 | 0 | 0 | 0 | 0 |
| 01 | 0 | 0 | 0 | 0 |
| 11 | 1 | 1 | 1 | 0 |
| 10 | 0 | 0 | 1 | 0 |

- 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