Logika cyfrowa

Wykład 2: optymalizacja układów kombinacyjnych

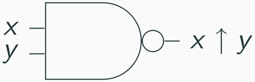
Marek Materzok

4 marca 2024

Spójniki zupełne

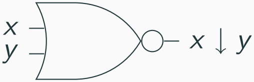
NAND i NOR

nand



i_1	i_2	o
0	0	1
0	1	1
1	0	1
1	1	0

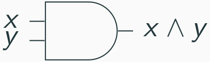
nor



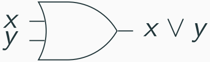
i_1	i_2	o
0	0	1
0	1	0
1	0	0
1	1	0

NAND i NOR

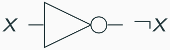
AND



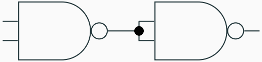
OR



NOT



$(x \uparrow y) \uparrow (x \uparrow y)$

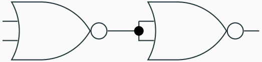


?

$x \uparrow x$

?

$(x \downarrow y) \downarrow (x \downarrow y)$



$x \downarrow x$

Postaci normalne

Negacyjna postać normalna

Formuła jest w *negacyjnej postaci normalnej* (NNF), jeśli negacja występuje w niej tylko bezpośrednio przy zmiennych.

Formułę można przekształcić do NNF używając praw de Morgana.

Formuły nie w NNF: $\neg(x \wedge \neg y) \vee z$, $x \wedge \neg(y \wedge (x \vee z))$

Formuły w NNF: $\neg x \vee y \vee z$, $x \wedge (\neg y \vee \neg x \wedge \neg z)$

Koniunkcyjna postać normalna

- *Literał* – zanegowana lub niezanegowana zmienna ($\neg x$ lub x)
- *Klauzula* – alternatywa literałów

Formuła jest w *koniuncyjnej postaci normalnej* (CNF), jeśli jest koniunkcją klauzul. Inaczej: iloczyn sum (product of sums, PoS).

Formułę w NNF można przekształcić do CNF używając prawa rozdzielności.

Formuły w NNF: $\neg(x \wedge \neg y) \vee z$, $x \wedge (\neg y \vee \neg x \wedge \neg z)$

Formuły w CNF: $\neg x \vee y \vee z$, $x \wedge (\neg y \vee \neg x) \wedge (\neg y \vee \neg z)$

Czytelniej: $\bar{x} + y + z$, $x(\bar{y} + \bar{x})(\bar{y} + \bar{z})$

Dysjunkcyjna postać normalna

Formuła jest w *dysjunkcyjnej postaci normalnej* (DNF), jeśli jest alternatywą koniunkcji literałów. Inaczej: suma iloczynów (sum of products, SoP).

Formułę w NNF można przekształcić do DNF używając prawa rozdzielności.

Formuły w NNF: $\neg(x \wedge \neg y) \vee z$, $x \wedge (\neg y \vee \neg x \wedge \neg z)$

Formuły w DNF: $\neg x \vee y \vee z$, $x \wedge \neg y \vee x \wedge \neg x \wedge \neg z$

Czytelniej: $\bar{x} + y + z$, $x\bar{y} + x\bar{x}\bar{z}$

Kanoniczna dysjunkcyjna postać normalna

- *Minterm* – koniunkcja literałów, w której każda rozważana zmienna występuje dokładnie raz

Formuła jest w *kanonicznej dysjunkcyjnej postaci normalnej* (CDNF), jeśli jest alternatywą mintermów. Inaczej: kanoniczna suma iloczynów.

Formuły w CNF: $\bar{x} + y + z$, $x\bar{y} + x\bar{x}\bar{z}$

Formuły w CDNF:
$xyz + xy\bar{z} + x\bar{y}z + \bar{x}yz + \bar{x}y\bar{z} + \bar{x}\bar{y}z + \bar{x}\bar{y}\bar{z}$
$x\bar{y}z + x\bar{y}\bar{z}$

Kanoniczna koniunkcyjna postać normalna

- *Maxterm* – dysjunkcja literałów, w której każda rozważana zmienna występuje dokładnie raz

Formuła jest w *kanonicznej koniunkcyjnej postaci normalnej* (CCNF), jeśli jest koniunkcją maxtermów. Inaczej: kanoniczny iloczyn sum.

Formuły w DNF: $\bar{x} + y + z$, $x(\bar{y} + \bar{x})(\bar{y} + \bar{z})$

Formuły w CCNF:

$\bar{x} + y + z$

$(x + y + z)(x + y + \bar{z})(x + \bar{y} + z)(x + \bar{y} + \bar{z})(\bar{x} + \bar{y} + z)(\bar{x} + \bar{y} + \bar{z})$

Numeracja mintermów i maxtermów

nr	x	y	z	minterm	maxterm
0	0	0	0	$m_0 = \bar{x}\bar{y}\bar{z}$	$M_0 = x + y + z$
1	0	0	1	$m_1 = \bar{x}\bar{y}z$	$M_1 = x + y + \bar{z}$
2	0	1	0	$m_2 = \bar{x}y\bar{z}$	$M_2 = x + \bar{y} + z$
3	0	1	1	$m_3 = \bar{x}yz$	$M_3 = x + \bar{y} + \bar{z}$
4	1	0	0	$m_4 = x\bar{y}\bar{z}$	$M_4 = \bar{x} + y + z$
5	1	0	1	$m_5 = x\bar{y}z$	$M_5 = \bar{x} + y + \bar{z}$
6	1	1	0	$m_6 = xy\bar{z}$	$M_6 = \bar{x} + \bar{y} + z$
7	1	1	1	$m_7 = xyz$	$M_7 = \bar{x} + \bar{y} + \bar{z}$

Numeracja mintermów i maxtermów – przykłady

CDNF

$$xyz + xy\bar{z} + x\bar{y}z + \bar{x}yz + \bar{x}y\bar{z} + \bar{x}\bar{y}z + \bar{x}\bar{y}\bar{z}$$
$$= m_0 + m_1 + m_2 + m_3 + m_5 + m_6 + m_7 = \sum m(0, 1, 2, 3, 5, 6, 7)$$

$$x\bar{y}z + x\bar{y}\bar{z}$$
$$= m_4 + m_5 = \sum m(4, 5)$$

CCNF

$$\bar{x} + y + z$$
$$= M_4 = \prod M(4)$$

$$(x + y + z)(x + y + \bar{z})(x + \bar{y} + z)(x + \bar{y} + \bar{z})(\bar{x} + \bar{y} + z)(\bar{x} + \bar{y} + \bar{z})$$
$$= M_0 M_1 M_2 M_3 M_6 M_7 = \prod M(0, 1, 2, 3, 6, 7)$$

Postaci kanoniczne a tabelki logiczne

nr	x	y	z	f(x,y,z)
0	0	0	0	0
1	0	0	1	1
2	0	1	0	0
3	0	1	1	1
4	1	0	0	1
5	1	0	1	0
6	1	1	0	0
7	1	1	1	0

Postaci kanoniczne a tabelki logiczne

nr	x	y	z	f(x,y,z)
0	0	0	0	0
1	0	0	1	1
2	0	1	0	0
3	0	1	1	1
4	1	0	0	1
5	1	0	1	0
6	1	1	0	0
7	1	1	1	0

Postaci kanoniczne a tabelki logiczne

nr	x	y	z	f(x,y,z)
0	0	0	0	0
1	0	0	1	1
2	0	1	0	0
3	0	1	1	1
4	1	0	0	1
5	1	0	1	0
6	1	1	0	0
7	1	1	1	0

$$f(x, y, z) = \bar{x}\bar{y}z + \bar{x}yz + x\bar{y}\bar{z} = \sum m(1, 3, 4)$$

Postaci kanoniczne a tabelki logiczne

nr	x	y	z	f(x,y,z)
0	0	0	0	0
1	0	0	1	1
2	0	1	0	0
3	0	1	1	1
4	1	0	0	1
5	1	0	1	0
6	1	1	0	0
7	1	1	1	0

$$f(x,y,z) = \bar{x}\bar{y}z + \bar{x}yz + x\bar{y}\bar{z} = \sum m(1,3,4)$$

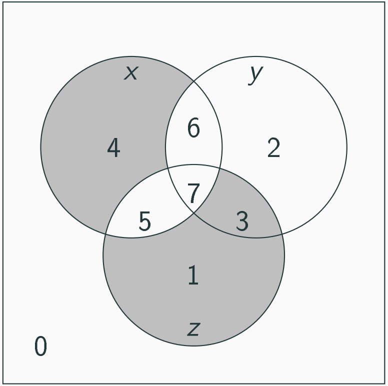
Postaci kanoniczne a tabelki logiczne

nr	x	y	z	f(x,y,z)
0	0	0	0	0
1	0	0	1	1
2	0	1	0	0
3	0	1	1	1
4	1	0	0	1
5	1	0	1	0
6	1	1	0	0
7	1	1	1	0

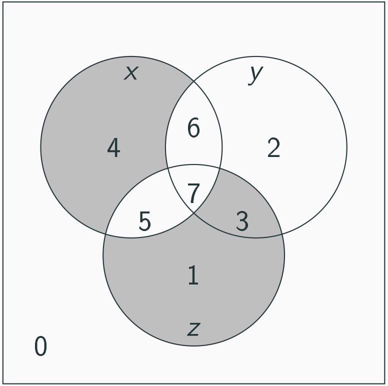
$$f(x,y,z) = \bar{x}\bar{y}z + \bar{x}yz + x\bar{y}\bar{z} = \sum m(1,3,4)$$

$$f(x,y,z) = (x+y+z)(x+\bar{y}+z)(\bar{x}+y+\bar{z})(\bar{x}+\bar{y}+z)(\bar{x}+\bar{y}+\bar{z}) = \prod M(0,2,5,6,7)$$

Postaci kanoniczne a diagramy Venna

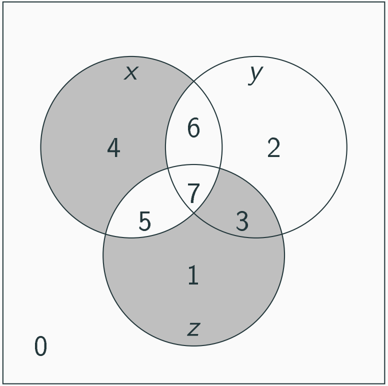


Postaci kanoniczne a diagramy Venna



$$f(x, y, z) = \sum m(1, 3, 4)$$

Postaci kanoniczne a diagramy Venna



$$f(x, y, z) = \sum m(1, 3, 4)$$
$$f(x, y, z) = \prod M(0, 2, 5, 6, 7)$$

Upraszczanie postaci kanonicznych

Używamy praw algebry Boole'a.

$$f(x, y, z) = \bar{x}\bar{y}z + \bar{x}yz + x\bar{y}\bar{z}$$

Upraszczanie postaci kanonicznych

Używamy praw algebry Boole'a.

$$
\begin{aligned}
f(x, y, z) &= \bar{x}\bar{y}z + \bar{x}yz + x\bar{y}\bar{z} \\
&= \bar{x}z(\bar{y} + y) + x\bar{y}\bar{z}
\end{aligned}
$$

Upraszczanie postaci kanonicznych

Używamy praw algebry Boole'a.

$$
\begin{aligned}
f(x, y, z) &= \bar{x}\bar{y}z + \bar{x}yz + x\bar{y}\bar{z} \\
&= \bar{x}z(\bar{y} + y) + x\bar{y}\bar{z} \\
&= \bar{x}z + x\bar{y}\bar{z}
\end{aligned}
$$

Upraszczanie postaci kanonicznych

Używamy praw algebry Boole'a.

$$\begin{aligned} f(x,y,z) &= \bar{x}\bar{y}z + \bar{x}yz + x\bar{y}\bar{z} \\ &= \bar{x}z(\bar{y}+y) + x\bar{y}\bar{z} \\ &= \bar{x}z + x\bar{y}\bar{z} \end{aligned}$$

$$g(x,y,z) = (x+y+z)(x+y+\bar{z})(\bar{x}+y+\bar{z})$$

Upraszczanie postaci kanonicznych

Używamy praw algebry Boole'a.

$$
\begin{aligned}
f(x,y,z) &= \bar{x}\bar{y}z + \bar{x}yz + x\bar{y}\bar{z} \\
&= \bar{x}z(\bar{y}+y) + x\bar{y}\bar{z} \\
&= \bar{x}z + x\bar{y}\bar{z}
\end{aligned}
$$

$$
\begin{aligned}
g(x,y,z) &= (x+y+z)(x+y+\bar{z})(\bar{x}+y+\bar{z}) \\
&= (x+y+z)(x+y+\bar{z})(x+y+\bar{z})(\bar{x}+y+\bar{z})
\end{aligned}
$$

Upraszczanie postaci kanonicznych

Używamy praw algebry Boole'a.

$$
\begin{aligned}
f(x,y,z) &= \bar{x}\bar{y}z + \bar{x}yz + x\bar{y}\bar{z} \\
&= \bar{x}z(\bar{y}+y) + x\bar{y}\bar{z} \\
&= \bar{x}z + x\bar{y}\bar{z}
\end{aligned}
$$

$$
\begin{aligned}
g(x,y,z) &= (x+y+z)(x+y+\bar{z})(\bar{x}+y+\bar{z}) \\
&= (x+y+z)(x+y+\bar{z})(x+y+\bar{z})(\bar{x}+y+\bar{z}) \\
&= (x+y+z\bar{z})(x\bar{x}+y+\bar{z})
\end{aligned}
$$

Upraszczanie postaci kanonicznych

Używamy praw algebry Boole'a.

$$
\begin{aligned}
f(x,y,z) &= \bar{x}\bar{y}z + \bar{x}yz + x\bar{y}\bar{z} \\
&= \bar{x}z(\bar{y}+y) + x\bar{y}\bar{z} \\
&= \bar{x}z + x\bar{y}\bar{z}
\end{aligned}
$$

$$
\begin{aligned}
g(x,y,z) &= (x+y+z)(x+y+\bar{z})(\bar{x}+y+\bar{z}) \\
&= (x+y+z)(x+y+\bar{z})(x+y+\bar{z})(\bar{x}+y+\bar{z}) \\
&= (x+y+z\bar{z})(x\bar{x}+y+\bar{z}) \\
&= (x+y)(y+\bar{z})
\end{aligned}
$$

Mapy Karnaugh

Mapa Karnaugh

- Dwuwymiarowy zapis tabelki logicznej
- Sąsiednie komórki odpowiadają wartościowaniom różniącym się **jedną** zmienną
- Topologia torusa – skrajnie lewa/prawa kolumna, górny/dolny wiersz są sąsiednie
- Umożliwia szybkie skonstruowanie niewielkiej równoważnej formuły rachunku Boole'a

Mapa Karnaugh dla 2 zmiennych

x	y	
0	0	m_0
0	1	m_1
1	0	m_2
1	1	m_3

$$
\begin{array}{cc|c|c|}
 & & \multicolumn{2}{c}{y} \\
 & & 0 & 1 \\
\hline
\multirow{2}{*}{x} & 0 & m_0 & m_1 \\
\hline
 & 1 & m_2 & m_3 \\
\hline
\end{array}
$$

Mapa Karnaugh dla 3 zmiennych

x	y	z	
0	0	0	m_0
0	0	1	m_1
0	1	0	m_2
0	1	1	m_3
1	0	0	m_4
1	0	1	m_5
1	1	0	m_6
1	1	1	m_7

		yz			
		00	01	11	10
x	0	m_0	m_1	m_3	m_2
	1	m_4	m_5	m_7	m_6

Mapa Karnaugh dla 4 zmiennych

$xy \backslash zw$	00	01	11	10
00	m_0	m_1	m_3	m_2
01	m_4	m_5	m_7	m_6
11	m_{12}	m_{13}	m_{15}	m_{14}
10	m_8	m_9	m_{11}	m_{10}

Mapa Karnaugh – grupy

n	x	y	z	Φ
0	0	0	0	1
1	0	0	1	0
2	0	1	0	0
3	0	1	1	0
4	1	0	0	0
5	1	0	1	0
6	1	1	0	0
7	1	1	1	0

x \ yz	00	01	11	10
0	1	0	0	0
1	0	0	0	0

Mapa Karnaugh – grupy

n	x	y	z	Φ
0	0	0	0	1
1	0	0	1	0
2	0	1	0	0
3	0	1	1	0
4	1	0	0	0
5	1	0	1	0
6	1	1	0	0
7	1	1	1	0

$x \backslash yz$	00	01	11	10
0	$\boxed{1}$	0	0	0
1	0	0	0	0

$$\Phi = m_0 = \bar{x}\bar{y}\bar{z}$$

Mapa Karnaugh – grupy

n	x	y	z	Φ
0	0	0	0	1
1	0	0	1	1
2	0	1	0	0
3	0	1	1	0
4	1	0	0	0
5	1	0	1	0
6	1	1	0	0
7	1	1	1	0

x \ yz	00	01	11	10
0	1	1	0	0
1	0	0	0	0

$$\Phi = \bar{x}\bar{y}$$

Mapa Karnaugh – grupy

n	x	y	z	Φ
0	0	0	0	0
1	0	0	1	1
2	0	1	0	0
3	0	1	1	1
4	1	0	0	0
5	1	0	1	0
6	1	1	0	0
7	1	1	1	0

x \ yz	00	01	11	10
0	0	1	1	0
1	0	0	0	0

$$\Phi = \bar{x}z$$

Mapa Karnaugh – grupy

n	x	y	z	Φ
0	0	0	0	1
1	0	0	1	0
2	0	1	0	1
3	0	1	1	0
4	1	0	0	0
5	1	0	1	0
6	1	1	0	0
7	1	1	1	0

yz

x	00	01	11	10
0	1	0	0	1
1	0	0	0	0

$$\Phi = \bar{x}\bar{z}$$

Mapa Karnaugh – grupy

n	x	y	z	Φ
0	0	0	0	1
1	0	0	1	1
2	0	1	0	1
3	0	1	1	1
4	1	0	0	0
5	1	0	1	0
6	1	1	0	0
7	1	1	1	0

x \ yz	00	01	11	10
0	1	1	1	1
1	0	0	0	0

$$\Phi = \bar{x}$$

Mapa Karnaugh – grupy

n	x	y	z	Φ
0	0	0	0	1
1	0	0	1	1
2	0	1	0	0
3	0	1	1	0
4	1	0	0	1
5	1	0	1	1
6	1	1	0	0
7	1	1	1	0

x \ yz	00	01	11	10
0	1	1	0	0
1	1	1	0	0

$$\Phi = \bar{y}$$

Mapa Karnaugh – dwie grupy

n	x	y	z	Φ
0	0	0	0	1
1	0	0	1	0
2	0	1	0	1
3	0	1	1	0
4	1	0	0	0
5	1	0	1	1
6	1	1	0	0
7	1	1	1	1

x \ yz	00	01	11	10
0	1	0	0	1
1	0	1	1	0

Mapa Karnaugh – dwie grupy

n	x	y	z	Φ
0	0	0	0	1
1	0	0	1	0
2	0	1	0	1
3	0	1	1	0
4	1	0	0	0
5	1	0	1	1
6	1	1	0	0
7	1	1	1	1

x \ yz	00	01	11	10
0	1	0	0	1
1	0	1	1	0

Mapa Karnaugh – dwie grupy

n	x	y	z	Φ
0	0	0	0	1
1	0	0	1	0
2	0	1	0	1
3	0	1	1	0
4	1	0	0	0
5	1	0	1	1
6	1	1	0	0
7	1	1	1	1

yz

x	00	01	11	10
0	1	0	0	1
1	0	1	1	0

$$\Phi = \bar{x}\bar{z} + xz$$

Mapa Karnaugh – dwie grupy (nieoptymalne!)

n	x	y	z	Φ
0	0	0	0	1
1	0	0	1	1
2	0	1	0	0
3	0	1	1	0
4	1	0	0	0
5	1	0	1	1
6	1	1	0	0
7	1	1	1	0

yz

x	00	01	11	10
0	1	1	0	0
1	0	1	0	0

$$\Phi = \bar{x}\bar{y} + x\bar{y}z$$

Mapa Karnaugh – nakładające się grupy

n	x	y	z	Φ
0	0	0	0	1
1	0	0	1	1
2	0	1	0	0
3	0	1	1	0
4	1	0	0	0
5	1	0	1	1
6	1	1	0	0
7	1	1	1	0

x \ yz	00	01	11	10
0	1	1	0	0
1	0	1	0	0

$$\Phi = \bar{x}\bar{y} + \bar{y}z$$

Mapa Karnaugh – grupy zer

n	x	y	z	Φ
0	0	0	0	0
1	0	0	1	0
2	0	1	0	1
3	0	1	1	1
4	1	0	0	1
5	1	0	1	0
6	1	1	0	1
7	1	1	1	1

x \ yz	00	01	11	10
0	0	0	1	1
1	1	0	1	1

$$\Phi = (x + y)(y + \bar{z})$$

Mapa Karnaugh – don't care

n	x	y	z	Φ
0	0	0	0	1
1	0	0	1	1
2	0	1	0	0
3	0	1	1	0
4	1	0	0	x
5	1	0	1	1
6	1	1	0	0
7	1	1	1	0

x \ yz	00	01	11	10
0	1	1	0	0
1	-	1	0	0

$$\Phi = \bar{x}\bar{y} + \bar{y}z$$

Mapa Karnaugh – don't care

n	x	y	z	Φ
0	0	0	0	1
1	0	0	1	1
2	0	1	0	0
3	0	1	1	0
4	1	0	0	x
5	1	0	1	1
6	1	1	0	0
7	1	1	1	0

x \ yz	00	01	11	10
0	1	1	0	0
1	-	1	0	0

$$\Phi = \bar{y}$$

Większy przykład

		zw			
		00	01	11	10
xy	00	1	0	0	1
	01	0	0	0	0
	11	1	1	1	0
	10	1	1	0	1

$$\Phi =$$

Większy przykład

xy \ zw	00	01	11	10
00	1	0	0	1
01	0	0	0	0
11	1	1	1	0
10	1	1	0	1

$$\Phi = x\bar{z} +$$

Większy przykład

zw

xy \ zw	00	01	11	10
00	1	0	0	1
01	0	0	0	0
11	1	1	1	0
10	1	1	0	1

$$\Phi = x\bar{z} + xyw +$$

Większy przykład

$xy \backslash zw$	00	01	11	10
00	1	0	0	1
01	0	0	0	0
11	1	1	1	0
10	1	1	0	1

$$\Phi = x\bar{z} + xyw + \bar{y}\bar{w}$$

Glitche

Glitch

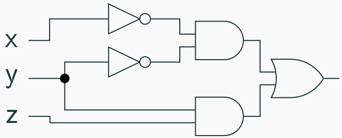
- W układach kombinacyjnych, gdy zmiana jednego wejścia wywołuje kilka zmian wyjścia
- Wywołane przez różnice w czasie propagacji pomiędzy ścieżkami w układzie
- W praktyce zależne od: technologii, różnic w produkcji, temperatury...

Glitch – przykład

$x = 0$, $z = 1$, y zmienia się z 1 na 0

x \ yz	00	01	11	10
0	1	1	1	0
1	0	0	1	0

$$\Phi = \bar{x}\bar{y} + yz$$



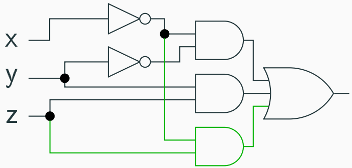

```
module glitch(output o, input x, y, z);
  assign o = !x && !y || y && z;
endmodule
```

Glitch – eliminacja

$x = 0$, $z = 1$, y zmienia się z 1 na 0

x \ yz	00	01	11	10
0	1	1	1	0
1	0	0	1	0

$$\Phi = \bar{x}\bar{y} + yz + \bar{x}z$$




```
module noglitch(output o, input x, y, z);
  assign o = !x && !y || y && z || !x && z;
endmodule
```

Więcej o glitchach

- W *układach synchronicznych* glitche nie stanowią problemu (patrz późniejsze wykłady)
- Glitche przy zmianach wielu wejść są praktycznie nie do uniknięcia
- Warto umieć **rozpoznać** glitch (w symulacji lub prawdziwym układzie)

System binarny

System pozycyjny

- Pozycja cyfry określa jej wagę.
- Doskonale znamy system dziesiętny (o podstawie 10).

$$2019 = (2019)_{10} = 2 \cdot 10^3 + 0 \cdot 10^2 + 1 \cdot 10^1 + 9 \cdot 10^0$$

System binarny

- System pozycyjny o podstawie 2.
- Dwie cyfry: 0, 1.
- Cyfra dwójkowa (binarna) to inaczej *bit*.

$$\begin{aligned}(1101)_2 &= 1 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 \\ &= 8 + 4 + 1 = 13\end{aligned}$$

Oznaczenia: `1101b`, `0b1101`

System ósemkowy

- System pozycyjny o podstawie 8.
- Osiem cyfr: 0, 1, 2, 3, 4, 5, 6, 7.
- Cyfra ósemkowa (oktalna) reprezentuje trzy bity.

$$
\begin{aligned}
(640)_8 &= 6 \cdot 8^2 + 4 \cdot 8^1 + 0 \cdot 8^0 = 6 \cdot 64 + 4 \cdot 8 = 416 \\
&= (110)_2 \cdot 8^2 + (100)_2 \cdot 8^1 + (000_2) \cdot 8^0 \\
&= (110100000)_2
\end{aligned}
$$

Oznaczenia: 640o, 0o640, 0640

System szesnastkowy

- System pozycyjny o podstawie 16.
- Szesnaście cyfr: $0 \ldots 9$, A, B, C, D, E, F.
- Cyfra szesnastkowa (heksadecymalna) reprezentuje cztery bity (inaczej *nibble*, *półbajt*)
- Dwie cyfry szesnastkowe tworzą *bajt* (byte, „gryz”)

$$
\begin{aligned}
(EA)_{16} &= 14 \cdot 16^1 + 10 \cdot 16^0 = 224 + 10 = 234 \\
&= (1110)_2 \cdot 16^1 + (1010)_2 \cdot 16^0 = \\
&= (11101010)_2
\end{aligned}
$$

Oznaczenia: `EAh`, `0xEA`, `#EA`

Potęgi dwójki

nr	dec	hex	oct	bin
0	1	1	1	1
1	2	2	2	10
2	4	4	4	100
3	8	8	10	1000
4	16	10	20	10000
5	32	20	40	100000
6	64	40	100	1000000
7	128	80	200	10000000
8	256	100	400	100000000
9	512	200	1000	1000000000
10	1024	400	2000	10000000000

Konwersja dziesiętny→binarny

Metoda 1:

- Znajdź największą potęgę 2 (np. 2^k) nie większą od n
- Oblicz $n' = n - 2^k$
- Jeśli n' większe od 0, powtórz dla n'
- Wynik posiada 1 na pozycjach zapisanych wykładników

$$133 =$$

Konwersja dziesiętny→binarny

Metoda 1:

- Znajdź największą potęgę 2 (np. 2^k) nie większą od n
- Oblicz $n' = n - 2^k$
- Jeśli n' większe od 0, powtórz dla n'
- Wynik posiada 1 na pozycjach zapisanych wykładników

$$133 = 128 + 5 =$$

Konwersja dziesiętny→binarny

Metoda 1:

- Znajdź największą potęgę 2 (np. 2^k) nie większą od n
- Oblicz $n' = n - 2^k$
- Jeśli n' większe od 0, powtórz dla n'
- Wynik posiada 1 na pozycjach zapisanych wykładników

$$133 = 128 + 5 = 128 + 4 + 1 =$$

Konwersja dziesiętny→binarny

Metoda 1:

- Znajdź największą potęgę 2 (np. 2^k) nie większą od n
- Oblicz $n' = n - 2^k$
- Jeśli n' większe od 0, powtórz dla n'
- Wynik posiada 1 na pozycjach zapisanych wykładników

$$133 = 128 + 5 = 128 + 4 + 1 = 2^7 + 2^2 + 2^0 = (10000101)_2$$

Konwersja dziesiętny→binarny

Metoda 1:

- Znajdź największą potęgę 2 (np. 2^k) nie większą od n
- Oblicz $n' = n - 2^k$
- Jeśli n' większe od 0, powtórz dla n'
- Wynik posiada 1 na pozycjach zapisanych wykładników

$$133 = 128 + 5 = 128 + 4 + 1 = 2^7 + 2^2 + 2^0 = (10000101)_2$$

Wynik jest budowany od bitów *najstarszych* (*najbardziej znaczących*, MSB).

Konwersja dziesiętny→binarny

Metoda 2:

- Podziel n przez 2 z resztą ($n = 2n' + r$)
- Dopisz resztę jako bit z lewej strony wyniku
- Jeśli n' większe od 0, powtórz dla n'

$$133 \quad =$$

Konwersja dziesiętny→binarny

Metoda 2:

- Podziel n przez 2 z resztą ($n = 2n' + r$)
- Dopisz resztę jako bit z lewej strony wyniku
- Jeśli n' większe od 0, powtórz dla n'

$$133 \quad = \quad 66 \cdot 2 + (1)_2 =$$

Konwersja dziesiętny→binarny

Metoda 2:

- Podziel n przez 2 z resztą ($n = 2n' + r$)
- Dopisz resztę jako bit z lewej strony wyniku
- Jeśli n' większe od 0, powtórz dla n'

$$133 \quad = \quad 66 \cdot 2 + (1)_2 = 33 \cdot 2^2 + (01)_2 =$$

Konwersja dziesiętny→binarny

Metoda 2:

- Podziel n przez 2 z resztą ($n = 2n' + r$)
- Dopisz resztę jako bit z lewej strony wyniku
- Jeśli n' większe od 0, powtórz dla n'

$$\begin{aligned} 133 &= 66 \cdot 2 + (1)_2 = 33 \cdot 2^2 + (01)_2 = 16 \cdot 2^3 + (101)_2 \\ &= \end{aligned}$$

Konwersja dziesiętny→binarny

Metoda 2:

- Podziel n przez 2 z resztą ($n = 2n' + r$)
- Dopisz resztę jako bit z lewej strony wyniku
- Jeśli n' większe od 0, powtórz dla n'

$$\begin{aligned} 133 &= 66 \cdot 2 + (1)_2 = 33 \cdot 2^2 + (01)_2 = 16 \cdot 2^3 + (101)_2 \\ &= 8 \cdot 2^4 + (0101)_2 = \end{aligned}$$

Konwersja dziesiętny→binarny

Metoda 2:

- Podziel n przez 2 z resztą ($n = 2n' + r$)
- Dopisz resztę jako bit z lewej strony wyniku
- Jeśli n' większe od 0, powtórz dla n'

$$
\begin{aligned}
133 &= 66 \cdot 2 + (1)_2 = 33 \cdot 2^2 + (01)_2 = 16 \cdot 2^3 + (101)_2 \\
&= 8 \cdot 2^4 + (0101)_2 = 4 \cdot 2^5 + (00101)_2 \\
&=
\end{aligned}
$$

Metoda 2:

- Podziel n przez 2 z resztą ($n = 2n' + r$)
- Dopisz resztę jako bit z lewej strony wyniku
- Jeśli n' większe od 0, powtórz dla n'

$$\begin{aligned}
133 &= 66 \cdot 2 + (1)_2 = 33 \cdot 2^2 + (01)_2 = 16 \cdot 2^3 + (101)_2 \\
&= 8 \cdot 2^4 + (0101)_2 = 4 \cdot 2^5 + (00101)_2 \\
&= 2 \cdot 2^6 + (000101)_2 =
\end{aligned}$$

Konwersja dziesiętny→binarny

Metoda 2:

- Podziel n przez 2 z resztą ($n = 2n' + r$)
- Dopisz resztę jako bit z lewej strony wyniku
- Jeśli n' większe od 0, powtórz dla n'

$$
\begin{aligned}
133 &= 66 \cdot 2 + (1)_2 = 33 \cdot 2^2 + (01)_2 = 16 \cdot 2^3 + (101)_2 \\
&= 8 \cdot 2^4 + (0101)_2 = 4 \cdot 2^5 + (00101)_2 \\
&= 2 \cdot 2^6 + (000101)_2 = 1 \cdot 2^7 + (0000101)_2 \\
&=
\end{aligned}
$$

Konwersja dziesiętny→binarny

Metoda 2:

- Podziel n przez 2 z resztą ($n = 2n' + r$)
- Dopisz resztę jako bit z lewej strony wyniku
- Jeśli n' większe od 0, powtórz dla n'

$$
\begin{aligned}
133 &= 66 \cdot 2 + (1)_2 = 33 \cdot 2^2 + (01)_2 = 16 \cdot 2^3 + (101)_2 \\
&= 8 \cdot 2^4 + (0101)_2 = 4 \cdot 2^5 + (00101)_2 \\
&= 2 \cdot 2^6 + (000101)_2 = 1 \cdot 2^7 + (0000101)_2 \\
&= (10000101)_2
\end{aligned}
$$

Konwersja dziesiętny→binarny

Metoda 2:

- Podziel n przez 2 z resztą ($n = 2n' + r$)
- Dopisz resztę jako bit z lewej strony wyniku
- Jeśli n' większe od 0, powtórz dla n'

$$\begin{aligned}
133 &= 66 \cdot 2 + (1)_2 = 33 \cdot 2^2 + (01)_2 = 16 \cdot 2^3 + (101)_2 \\
&= 8 \cdot 2^4 + (0101)_2 = 4 \cdot 2^5 + (00101)_2 \\
&= 2 \cdot 2^6 + (000101)_2 = 1 \cdot 2^7 + (0000101)_2 \\
&= (10000101)_2
\end{aligned}$$

Wynik jest budowany od bitów *najmłodszych* (*najmniej znaczących*, LSB).

Sygnały wielobitowe w SystemVerilogu

Stałe liczbowe bez wymiaru

Formaty stałych bez wymiaru:

- `42` – dziesiętna
- `'d42` – dziesiętna
- `'xFA` – szesnastkowa
- `'o17` – ósemkowa
- `'b1101` – binarna

Użycie stałych bez wymiaru do specyfikowania wartości sygnałów jest **niewskazane**!

Stałe liczbowe z wymiarem

Liczba przed apostrofem oznacza liczbę bitów:

- `8'd42` – dziesiętna
- `8'xFA` – szesnastkowa
- `6'o17` – ósemkowa
- `4'b1101` – binarna

Gdy zadeklarowana liczba bitów jest większa niż wynika z liczby cyfr, najstarsze bity są wypełniane zerami.

`8'b1101` oznacza `8'b00001101`.

Wektory bitowe

Zmienne wielobitowe.

Notacja: [msb:lsb], gdzie msb i lsb to numery najbardziej i najmniej znaczącego bitu.

```
module beef(output [15:0] o);
  assign o = 16'hbeef;
endmodule
```

Konkatenacja wektorów

Skleja bity składowych wektorów, bez modyfikacji, w jeden dłuższy wektor.

Notacja: `{w1, w2, ..., wn}`, gdzie `w1` do `wn` obliczają się do składowych wektorów.

```
module concat(output [15:0] o, input [7:0] i, j);
  assign o = { i, j };
endmodule
```

Konkatenacja grupuje sygnały, nie wprowadza nowych bramek.

Konkatenacja jako rozdzielanie

Konkatenacja może być użyta po lewej stronie przypisania – wtedy rozdziela wektor na fragmenty.

```
module split(output [7:0] o, p, input [15:0] i);
  assign {o, p} = i;
endmodule
```

Replikacja

Powtarza wielokrotnie bity wejściowego wektora.

Notacja: `{n{w}}`, gdzie `n` to stałe wyrażenie obliczające się do liczby powtórzeń, a `w` oblicza się do powtarzanego wektora.

```
module twice(output [15:0] o, input [7:0] i);
  assign o = {2{i}};
endmodule
```

Wybór bitu

Wybiera jeden bit z wektora bitowego. Numeracja zależy od numeracji bitów w wejściowym wektorze.

Notacja: `w[n]`, gdzie `w` oblicza się do wektora, a `n` oblicza się do numeru bitu (typowo stałego).

```
module fourth_lsb(output o, input [7:0] i);
  assign o = i[3];
endmodule

module fourth_msb(output o, input [0:7] i);
  assign o = i[3];
endmodule
```

Wybór podwektora

Wybiera podwektor z wektora bitowego.

Notacja: `w[msb:lsb]`, gdzie `w` oblicza się do wektora, a `msb` i `lsb` to stałe wyrażenia obliczające się do numerów najbardziej i najmniej znaczącego bitu wyniku.

```
module msnibble(output [3:0] o, input [7:0] i);
  assign o = i[7:4];
endmodule
```

Wybór podwektora – baza + rozmiar

Zamiast podawać numer pierwszego i ostatniego bitu, można podać numer pierwszego/ostatniego bitu i rozmiar wektora wynikowego.

Notacja: `w[n+:m]`, gdzie `w` oblicza się do wektora, `m` to stałe wyrażenie obliczajace się do rozmiaru wektora wynikowego, a `n` oblicza się do numeru najmniej/najbardziej znaczącego bitu początkowego (zależy od numeracji w `w`).

Notacja: `w[n-:m]` jest analogiczna, tylko liczy bity w przeciwnym kierunku.

```
module msnibble(output [3:0] o, input [7:0] i);
  assign o = i[4+:4]; // lub i[7-:4]
endmodule
```

Wybór podwektora dla przypisania fragmentu

Wybór podwektora może być użyty po lewej stronie przypisania. Wtedy pozwala na przypisanie do fragmentu wektora.

```
module swapnibbles(output [7:0] o, input [7:0] i);
  assign o[3:0] = i[7:4];
  assign o[7:4] = i[3:0];
endmodule
```

Operacje bitowe na wektorach

Można wykonać operację bitową równolegle na wszystkich bitach wektora, podobnie jak w C:

- Koniunkcja (AND) – `&`, NAND – `~&`
- Dysjunkcja (OR) – `|`, NOR – `~|`
- Alternatywa rozłączna (XOR) – `^`, XNOR – `~^`
- Negacja (NOT) – `~`

Wynikowy wektor ma tyle bitów, co **najdłuższy** wejściowy.

```
module bigxor(output [3:0] o, input [3:0] i, j);
  assign o = i ^ j;
endmodule
```

Operacje redukujące

Operatory binarne w SystemVerilogu można też użyć prefiksowo jako operatory unarne.

Oznacza to wtedy zastosowanie operatora do wszystkich bitów wejściowego wektora, wynik jest jednobitowy.

```
module andall(output o, input [3:0] i);
  assign o = &i;
endmodule
```