Logika cyfrowa

Wykład 4: podstawowe układy kombinacyjne

Marek Materzok

18 marca 2024

Multipleksery

Multiplekser

s	a_0	a_1	o
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

Multiplekser

s	a_0	a_1	o
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

s \ $a_0 a_1$	00	01	11	10
0	0	0	1	1
1	0	1	1	0

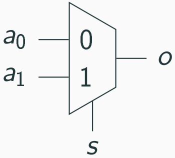
Multiplekser

s	a_0	a_1	o
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

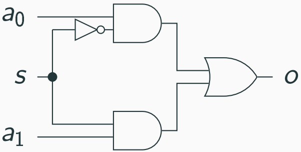
s \ $a_0 a_1$	00	01	11	10
0	0	0	1	1
1	0	1	1	0

Multiplekser

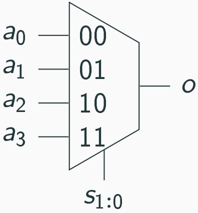
Obwód wybierający jedno z wejść:



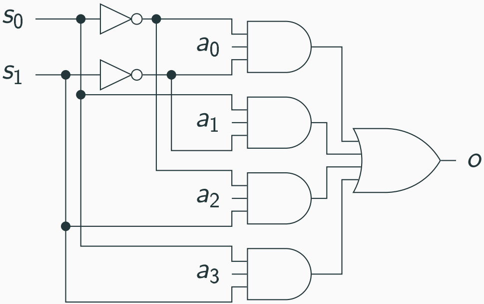
s	o
0	a_0
1	a_1



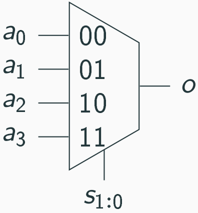
Multiplekser czterowejściowy



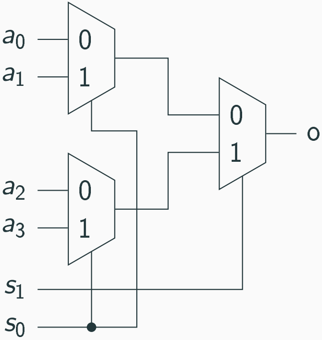
s_1	s_0	o
0	0	a_0
0	1	a_1
1	0	a_2
1	1	a_3



Multiplekser czterowejściowy

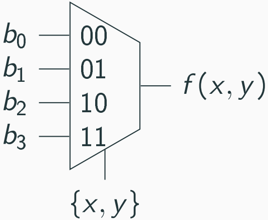


s_1	s_0	o
0	0	a_0
0	1	a_1
1	0	a_2
1	1	a_3



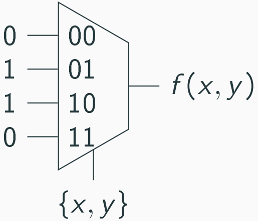
Implementacja funkcji logicznych

x	y	$f(x, y)$
0	0	b_0
0	1	b_1
1	0	b_2
1	1	b_3



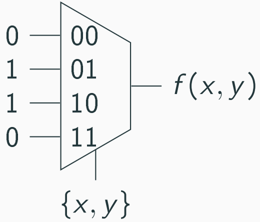
Implementacja funkcji logicznych – przykład

x	y	$f(x, y)$
0	0	0
0	1	1
1	0	1
1	1	0



Implementacja funkcji logicznych – przykład

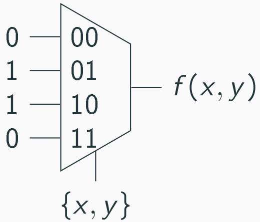
x	y	$f(x,y)$
0	0	0
0	1	1
1	0	1
1	1	0



Implementacja funkcji logicznych – przykład

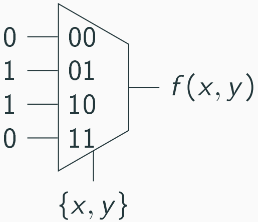
x	y	$f(x,y)$
0	0	0
0	1	1
1	0	1
1	1	0

x	$f(x,y)$
0	y
1	$\bar{y}$

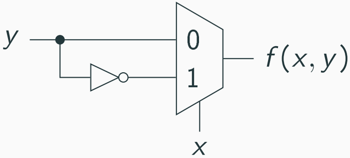


Implementacja funkcji logicznych – przykład

x	y	$f(x,y)$
0	0	0
0	1	1
1	0	1
1	1	0



x	$f(x,y)$
0	y
1	$\bar{y}$



Implementacja funkcji logicznych – większy przykład

x	y	z	o
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Implementacja funkcji logicznych – większy przykład

x	y	z	o
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Implementacja funkcji logicznych – większy przykład

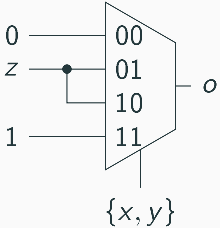
x	y	z	o
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

x	y	o
0	0	0
0	1	z
1	0	z
1	1	1

Implementacja funkcji logicznych – większy przykład

x	y	z	o
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

x	y	o
0	0	0
0	1	z
1	0	z
1	1	1



Rozwinięcie Shannona

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

Lub, dla formuł:

$$\Phi = x \wedge \Phi\{x/1\} \vee \neg x \wedge \Phi\{x/0\}$$

Rozwinięcie Shannona

$$f(x_1, x_2, \ldots x_n) = x_1 f(1, x_2, \ldots, x_n) + \bar{x_1} f(0, x_2, \ldots, x_n)$$

Lub, dla formuł:

$$\Phi = x \wedge \Phi\{x/1\} \vee \neg x \wedge \Phi\{x/0\}$$

Dualnie:

$$\begin{aligned} \Phi &= (x \vee \Phi\{x/0\}) \wedge (\neg x \vee \Phi\{x/1\}) \\ &= (\neg x \to \Phi\{x/0\}) \wedge (x \to \Phi\{x/1\}) \end{aligned}$$

Rozwinięcie Shannona

$$f(x_1, x_2, \ldots x_n) = x_1 f(1, x_2, \ldots, x_n) + \bar{x_1} f(0, x_2, \ldots, x_n)$$

Lub, dla formuł:

$$\Phi = x \wedge \Phi\{x/1\} \vee \neg x \wedge \Phi\{x/0\}$$

Dualnie:

$$\begin{aligned}\Phi &= (x \vee \Phi\{x/0\}) \wedge (\neg x \vee \Phi\{x/1\}) \\ &= (\neg x \rightarrow \Phi\{x/0\}) \wedge (x \rightarrow \Phi\{x/1\})\end{aligned}$$

Albo, językiem programistów:

$$\Phi = \texttt{if } x \texttt{ then } \Phi\{x/1\} \texttt{ else } \Phi\{x/0\}$$

Wyrażenie warunkowe w SystemVerilogu

Podobnie jak w języku C: `w ? a0 : a1`

Wynik wyrażenia `w` jest redukowany do jednego bitu.
Jeśli ten bit to 1, wynikiem jest wynik obliczenia a0.
Jeśli ten bit to 0, wynikiem jest wynik obliczenia a1.

Wynik wyrażenia warunkowego ma taką szerokość, co **większy** z wyników a0 i a1.
Mniejszy jest rozszerzany.

Wyrażenie warunkowe oznacza **multiplekser**.

Przykłady

XOR za pomocą multipleksera:

```
module my_xor(output o, input x, y);
  assign o = x ? ~y : y;
endmodule
```

2 lub 3 z trzech za pomocą multiplekserów:

```
module my_majority(output o, input x, y, z);
  assign o = x ? (y ? 1'b1 : z) : (y ? z : 1'b0);
endmodule
```

One-hot, dekodery i enkodery

Kodowanie one-hot

Kodujemy wartość zapaleniem **dokładnie jednego** z n bitów:

v	1hot
0	00000001
1	00000010
2	00000100
3	00001000
4	00010000
5	00100000
6	01000000
7	10000000

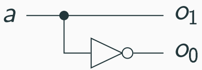
Dekoder

Konwertuje kod binarny na one-hot. Dekoder **1 do 2**:

a	o_1	o_0
0	0	1
1	1	0

$$o_0 = \bar{a}$$

$$o_1 = a$$



Dekoder 2 do 4

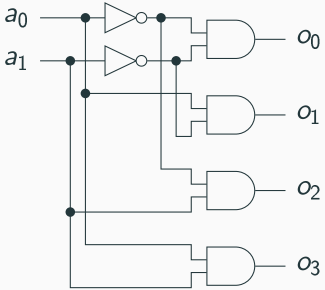
a_1	a_0	o_3	o_2	o_1	o_0
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0

$$o_0 = \bar{a}_1 \bar{a}_0$$

$$o_1 = \bar{a}_1 a_0$$

$$o_2 = a_1 \bar{a}_0$$

$$o_3 = a_1 a_0$$



Enkoder

Konwertuje kod one-hot na binarny. Enkoder **2 do 1**:

a_1	a_0	o	v
0	0	x	0
0	1	0	1
1	0	1	1
1	1	x	0

a_1 \ a_0	0	1
0	-	0
1	1	-

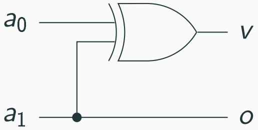
$o = a_1$

$v = a_0 \oplus a_1$

Enkoder

Konwertuje kod one-hot na binarny. Enkoder **2 do 1**:

a_1	a_0	o	v
0	0	x	0
0	1	0	1
1	0	1	1
1	1	x	0



$$o = a_1$$

$$v = a_0 \oplus a_1$$

Enkoder 4 do 2

$a_{3:0}$	o_1	o_0	v
0001	0	0	1
0010	0	1	1
0100	1	0	1
1000	1	1	1
w p.p.	x	x	0

$a_3 a_2$ \ $a_1 a_0$	00	01	11	10
00	-	0	-	1
01	0	-	-	-
11	-	-	-	-
10	1	-	-	-

Enkoder 4 do 2

$a_{3:0}$	o_1	o_0	v
0001	0	0	1
0010	0	1	1
0100	1	0	1
1000	1	1	1
w p.p.	x	x	0

$$o_0 = a_1 + a_3$$

$a_3 a_2$ \ $a_1 a_0$	00	01	11	10
00	-	0	-	1
01	0	-	-	-
11	-	-	-	-
10	1	-	-	-

Enkoder 4 do 2

$a_{3:0}$	o_1	o_0	v
0001	0	0	1
0010	0	1	1
0100	1	0	1
1000	1	1	1
w p.p.	x	x	0

$a_1 a_0$

$a_3 a_2$	00	01	11	10
00	-	0	-	0
01	1	-	-	-
11	-	-	-	-
10	1	-	-	-

$$o_0 = a_1 + a_3$$

Enkoder 4 do 2

$a_{3:0}$	o_1	o_0	v
0001	0	0	1
0010	0	1	1
0100	1	0	1
1000	1	1	1
w p.p.	x	x	0

$a_3 a_2$ \ $a_1 a_0$	00	01	11	10
00	-	0	-	0
01	1	-	-	-
11	-	-	-	-
10	1	-	-	-

$$o_0 = a_1 + a_3$$

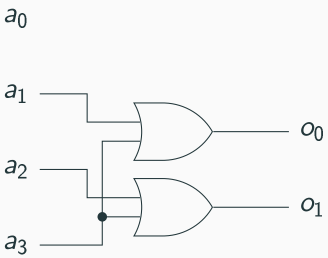
$$o_1 = a_2 + a_3$$

Enkoder 4 do 2

$a_{3:0}$	o_1	o_0	v
0001	0	0	1
0010	0	1	1
0100	1	0	1
1000	1	1	1
w p.p.	x	x	0

$$o_0 = a_1 + a_3$$

$$o_1 = a_2 + a_3$$



Enkoder priorytetowy

Oblicza numer najstarszej jedynki. Enkoder priorytetowy **2 do 1**:

a_1	a_0	o	v
0	0	x	0
0	1	0	1
1	0	1	1
1	1	1	1

		a_0	
		0	1
a_1	0	-	0
	1	1	1

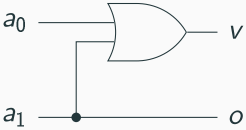
$$o = a_1$$

$$v = a_0 + a_1$$

Enkoder priorytetowy

Oblicza numer najstarszej jedynki. Enkoder priorytetowy **2 do 1**:

a_1	a_0	o	v
0	0	x	0
0	1	0	1
1	0	1	1
1	1	1	1



$$o = a_1$$

$$v = a_0 + a_1$$

Enkoder priorytetowy 4 do 2

$a_{3:0}$	o_1	o_0	v
0001	0	0	1
001x	0	1	1
01xx	1	0	1
1xxx	1	1	1
0000	x	x	0

$a_1 a_0$

$a_3 a_2$	00	01	11	10
00	-	0	1	1
01	0	0	0	0
11	1	1	1	1
10	1	1	1	1

Enkoder priorytetowy 4 do 2

$a_{3:0}$	o_1	o_0	v
0001	0	0	1
001x	0	1	1
01xx	1	0	1
1xxx	1	1	1
0000	x	x	0

$$o_0 = a_1 \bar{a}_2 + a_3$$

$a_3 a_2$ \ $a_1 a_0$	00	01	11	10
00	-	0	1	1
01	0	0	0	0
11	1	1	1	1
10	1	1	1	1

Enkoder priorytetowy 4 do 2

$a_{3:0}$	o_1	o_0	v
0001	0	0	1
001x	0	1	1
01xx	1	0	1
1xxx	1	1	1
0000	x	x	0

$a_3 a_2$ \ $a_1 a_0$	00	01	11	10
00	-	0	0	0
01	1	1	1	1
11	1	1	1	1
10	1	1	1	1

$$o_0 = a_1 \bar{a}_2 + a_3$$

Enkoder priorytetowy 4 do 2

$a_{3:0}$	o_1	o_0	v
0001	0	0	1
001x	0	1	1
01xx	1	0	1
1xxx	1	1	1
0000	x	x	0

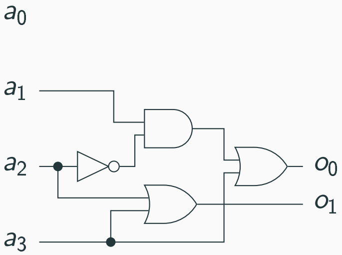
$$o_0 = a_1 \bar{a}_2 + a_3$$

$$o_1 = a_2 + a_3$$

$a_3 a_2$ \ $a_1 a_0$	00	01	11	10
00	-	0	0	0
01	1	1	1	1
11	1	1	1	1
10	1	1	1	1

Enkoder priorytetowy 4 do 2

$a_{3:0}$	o_1	o_0	v
0001	0	0	1
001x	0	1	1
01xx	1	0	1
1xxx	1	1	1
0000	x	x	0



$$o_0 = a_1 \bar{a}_2 + a_3$$

$$o_1 = a_2 + a_3$$

$$v = a_0 + a_1 + a_2 + a_3$$

Enkoder one-hot a enkoder priorytetowy

- Enkoder one-hot **dużo prostszy** od enkodera priorytetowego:
    - mniej bramek
    - krótsza ścieżka krytyczna
- W sytuacji, gdy możliwy jest wybór, należy **unikać** logiki priorytetowej
- Tłumacząc na język programistów: długie ciągi if-else **niewskazane**!

Multiplekser one-hot

s_1	s_0	a_0	a_1	o
0	0	0	0	x
0	0	0	1	x
0	0	1	0	x
0	0	1	1	x
0	1	0	0	0
0	1	0	1	0
0	1	1	0	1
0	1	1	1	1
1	0	0	0	0
1	0	0	1	1
1	0	1	0	0
1	0	1	1	1
1	1	0	0	x
1	1	0	1	x
1	1	1	0	x
1	1	1	1	x

Multiplekser one-hot

s_1	s_0	a_0	a_1	o
0	0	0	0	x
0	0	0	1	x
0	0	1	0	x
0	0	1	1	x
0	1	0	0	0
0	1	0	1	0
0	1	1	0	1
0	1	1	1	1
1	0	0	0	0
1	0	0	1	1
1	0	1	0	0
1	0	1	1	1
1	1	0	0	x
1	1	0	1	x
1	1	1	0	x
1	1	1	1	x

$s_1 s_0$ \ $a_0 a_1$	00	01	11	10
00	-	-	-	-
01	0	0	1	1
11	-	-	-	-
10	0	1	1	0

Multiplekser one-hot

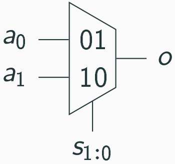
s_1	s_0	a_0	a_1	o
0	0	0	0	x
0	0	0	1	x
0	0	1	0	x
0	0	1	1	x
0	1	0	0	0
0	1	0	1	0
0	1	1	0	1
0	1	1	1	1
1	0	0	0	0
1	0	0	1	1
1	0	1	0	0
1	0	1	1	1
1	1	0	0	x
1	1	0	1	x
1	1	1	0	x
1	1	1	1	x

$a_0 a_1$

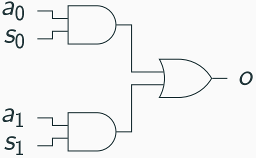
$s_1 s_0$	00	01	11	10
00	-	-	-	-
01	0	0	1	1
11	-	-	-	-
10	0	1	1	0

$$s_0 a_0 + s_1 a_1$$

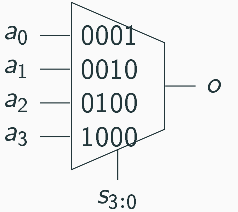
Multiplekser one-hot



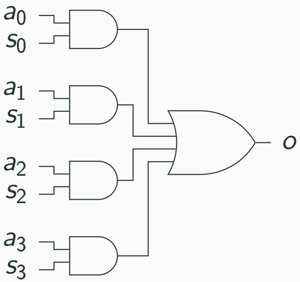
s_1	s_0	o
0	0	x
0	1	a_0
1	0	a_1
1	1	x



Multiplekser one-hot czterowejściowy

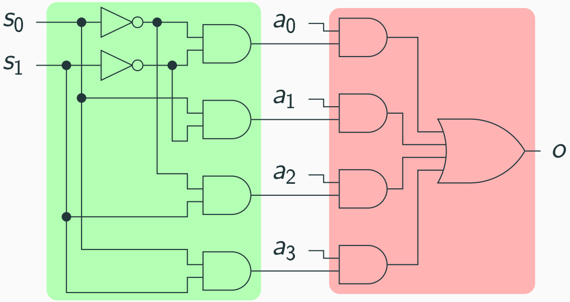


s_3	s_2	s_1	s_0	o
0	0	0	1	a_0
0	0	1	0	a_1
0	1	0	0	a_2
1	0	0	0	a_3
w p.p.				x



Multiplekser: konstrukcja

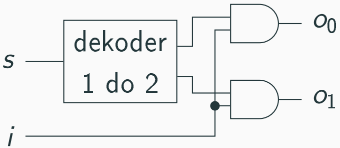
Dekoder + multiplekser one-hot:



Demultiplekser

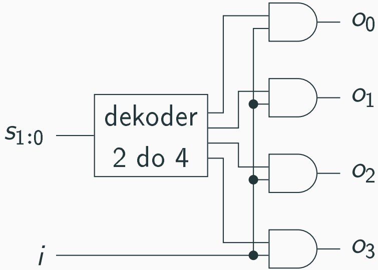
Przesyła wejście na jedno z wielu wyjść, pozostałe zeruje.

s	i	o_1	o_0
0	0	0	0
0	1	0	1
1	0	0	0
1	1	1	0



Demultiplekser czterowyjściowy

s_1	s_0	o_3	o_2	o_1	o_0
0	0	0	0	0	i
0	1	0	0	i	0
1	0	0	i	0	0
1	1	i	0	0	0



Konwertery kodów

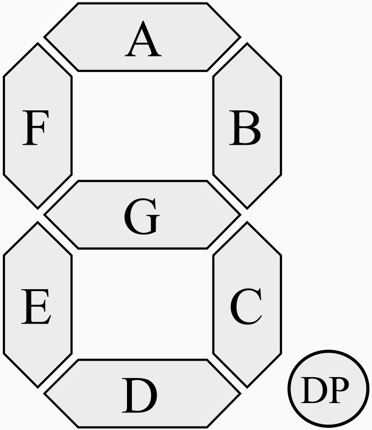
Enkoder/dekoder BCD

- Enkoder BCD: 10 do 4
  np. 40147
- Dekoder BCD: 4 do 10
  np. 4028, 74LS42
- Zastosowania: obsługa klawiatur i wyświetlaczy numerycznych

$a_{9:0}$	$b_{3:0}$
0000000001	0000
0000000010	0001
0000000100	0010
0000001000	0011
0000010000	0100
0000100000	0101
0001000000	0110
0010000000	0111
0100000000	1000
1000000000	1001

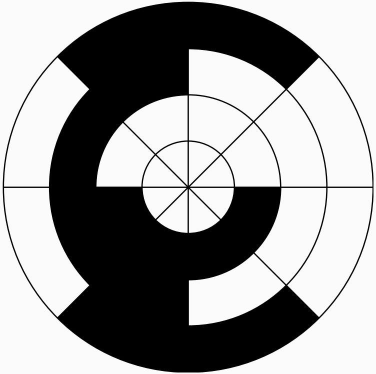
Wyświetlacz 7-segmentowy

n	g	f	e	d	c	b	a
0	0	1	1	1	1	1	1
1	0	0	0	0	1	1	0
2	1	0	1	1	0	1	1
3	1	0	0	1	1	1	1
4	1	1	0	0	1	1	0
5	1	1	0	1	1	0	1
6	1	1	1	1	1	0	1
7	0	0	0	0	1	1	1
8	1	1	1	1	1	1	1
9	1	1	0	1	1	1	1



Kody Graya

- Kolejne kody różnią się **dokładnie jednym** bitem
- Zastosowania: czujniki pozycji (enkodery), korekcja błędów



n	bin	Gray
0	000	000
1	001	001
2	010	011
3	011	010
4	100	110
5	101	111
6	110	101
7	111	100

Konstrukcja kodów Graya

Metoda odbij-i-doklej-bit:

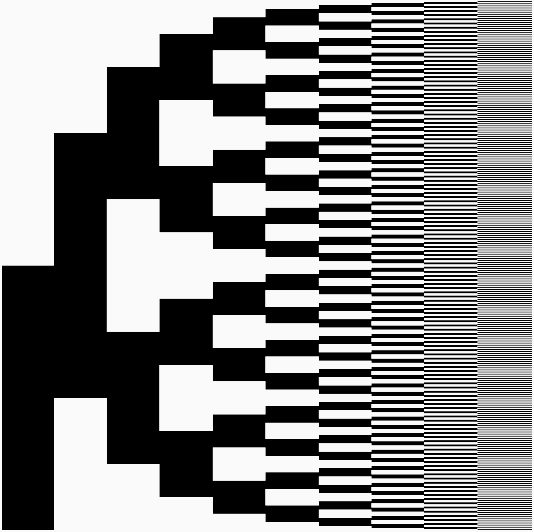
0	1	2	3
	0	00	000
	1	01	001
		11	011
		10	010
			110
			111
			101
			100

$$G(\epsilon) = \epsilon$$
$$G(0w) = 0G(w)$$
$$G(1w) = 1G(\bar{w})$$

Albo:

$$G(0) = 0$$
$$G(n) = 2^{\mathtt{MSB}(n)} + G(2^{\mathtt{MSB}(n)+1} - n - 1)$$

Kody Graya



Obliczanie kodów Graya

Obliczanie i-tego kodu Graya:

$$G(i) = i \oplus (i >> 1)$$

Obliczanie numeru n-bitowego kodu Graya:

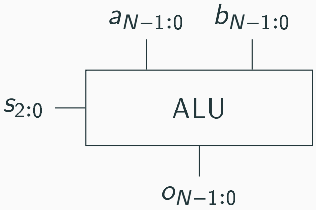
$$f_j(x) = x \oplus (x >> 2^j)$$
$$G^{-1}(i) = (f_0 \circ f_1 \circ \cdots \circ f_{\mathtt{MSB}(n-1)})(i)$$

ALU

Jednostka arytmetyczno-logiczna (ALU)

Układ realizujący wiele funkcji arytmetycznych i logicznych. Przykład:

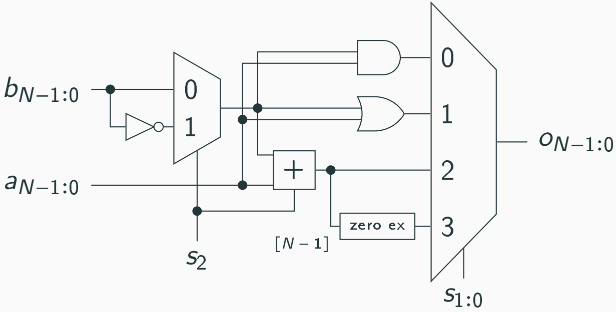
$f_{2:0}$	funkcja
000	$a \& b$
001	$a \mid b$
010	$a + b$
011	-
100	$a \& \bar{b}$
101	$a \mid \bar{b}$
110	$a - b$
111	SLT



Jednostka arytmetyczno-logiczna (ALU)

Układ realizujący wiele funkcji arytmetycznych i logicznych. Przykład:

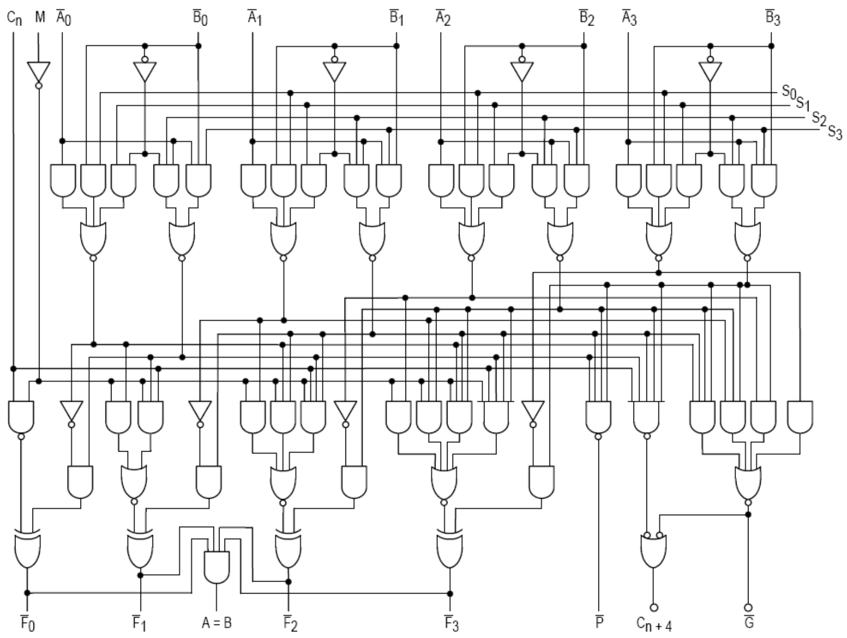
$f_{2:0}$	funkcja
000	$a \& b$
001	$a \mid b$
010	$a + b$
011	-
100	$a \& \bar{b}$
101	$a \mid \bar{b}$
110	$a - b$
111	SLT



Klasyczne ALU 74181

- Produkowany od lat 60
- Arytmetyka 4-bitowa
- Przewidywanie przeniesienia
- Przewidywanie hierarchiczne z układem 74182
- Używany w komputerach:
  - PDP-11 (1970-80)
  - VAX-11 (1977-84)
  - Xerox Alto (1973-81)
- Wciąż można go kupić! (z Chin)

$s_{3:0}$	$m = 1$	$m = 0$
0000	$\bar{a}$	$a - 1$
0001	$\overline{a \& b}$	$(a \& b) - 1)$
0010	$\bar{a} \mid b$	$(a \& \bar{b}) - 1$
0011	1	-1
0100	$\overline{a \mid b}$	$a + (a \mid \bar{b})$
0101	$\bar{b}$	$(a \& b) + (a \mid \bar{b})$
0110	$\overline{a \oplus b}$	$a - b - 1$
0111	$a \mid \bar{b}$	$a \mid \bar{b}$
1000	$\bar{a} \& b$	$a + (a \mid b)$
1001	$a \oplus b$	$a + b$
1010	b	$(a \& \bar{b}) + (a \mid b)$
1011	$a \mid b$	$a \mid b$
1100	0	$a + a$
1101	$a \& \bar{b}$	$(a \& b) + a$
1110	$a \& b$	$(a \& \bar{b}) + a$
1111	a	a



C_n
M
$\overline{A}_0$
$\overline{B}_0$
$\overline{A}_1$
$\overline{B}_1$
$\overline{A}_2$
$\overline{B}_2$
$\overline{A}_3$
$\overline{B}_3$
S_0
S_1
S_2
S_3
$\overline{F}_0$
$\overline{F}_1$
A = B
$\overline{F}_2$
$\overline{F}_3$
$\overline{P}$
C_{n+4}
$\overline{G}$