SUMATORY RÓWNOLEGŁE

Zamienność dodawania i odejmowania

W systemie pozycyjnym i uzupełnieniowym używając dopełnień
można zastąpić odejmowanie dodawaniem i *vice versa*:

$$\mathbf{X} - \mathbf{Y} = \overline{\overline{\mathbf{X}} + \mathbf{Y}}$$
$$\mathbf{X} + \mathbf{Y} = \overline{\overline{\mathbf{X}} - \mathbf{Y}} = \overline{\overline{\mathbf{Y}} - \mathbf{X}}$$

Zależność ta jest też widoczna w logicznych strukturach dodawania i odejmowania

Ponieważ $\mathbf{0} - \mathbf{Y} = \overline{\mathbf{Y}} + ulp$, gdzie $ulp=\beta^{-m}$ (o reprezentacji {0,...,0,1}), więc mamy także

$$\mathbf{X} - \mathbf{Y} = \mathbf{X} + \underline{\mathbf{Y}} = \mathbf{X} + \overline{\mathbf{Y}} + ulp$$
$$\mathbf{X} + \mathbf{Y} = \mathbf{X} - \underline{\mathbf{Y}} = \mathbf{X} - \overline{\mathbf{Y}} - ulp =$$
$$= \mathbf{Y} - \underline{\mathbf{X}} = \mathbf{Y} - \overline{\mathbf{X}} - ulp$$

Możliwe jest więc utworzenie uniwersalnego układu dodająco-odejmującego.

Logika dodawania i odejmowania

Logika dodawania i odejmowania pozycyjnego:

Dla danych $x_i, y_i \in \{0,1,\ldots,\beta-1\}, c_i \in \{0,1\}$ istnieją $s_i \in \{0,1,\ldots,\beta-1\}, c_{i+1} \in \{0,1\}$ takie, że

$$\begin{aligned} x_i + y_i + c_i &= \beta c_{i+1} + s_i, \\ x_i - y_i - c_i &= -\beta c_{i+1} + s_i, \end{aligned} \quad \text{gdzie } x_i, y_i, s_i \in \{0,1,\ldots,\beta-1\}, c_i \in \{0,1\},$$

czemu odpowiada iteracyjne (kaskadowe) powiązanie pozycji.

W podstawie β=2 równaniom arytmetycznym odpowiadają funkcje logiczne:

dodawanie $\overline{X+Y} = \overline{X} - Y = \overline{Y} - X$

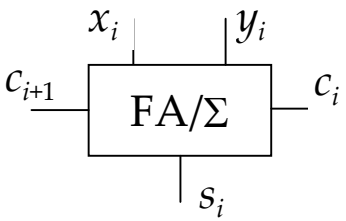
$$s_i = x_i \oplus y_i \oplus c_i = h_i \oplus c_i$$

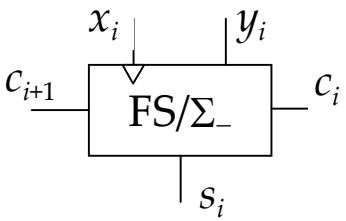
sumator:
$$\begin{aligned} c_{i+1} &= x_i y_i + (x_i \oplus y_i)\, c_i = \\ &= x_i y_i + (x_i + y_i)\, c_i = g_i + p_i c_i \end{aligned}$$

odejmowanie $\overline{X-Y} = \overline{X} + Y$

$$\bar{s}_i = \bar{x}_i \oplus y_i \oplus c_i = h_i \oplus c_i$$

subtraktor:
$$\begin{aligned} c_{i+1} &= \bar{x}_i y_i + (\bar{x}_i \oplus y_i)\, c_i = \\ &= \bar{x}_i y_i + (\bar{x}_i + y_i)\, c_i = g_i + p_i c_i \end{aligned}$$

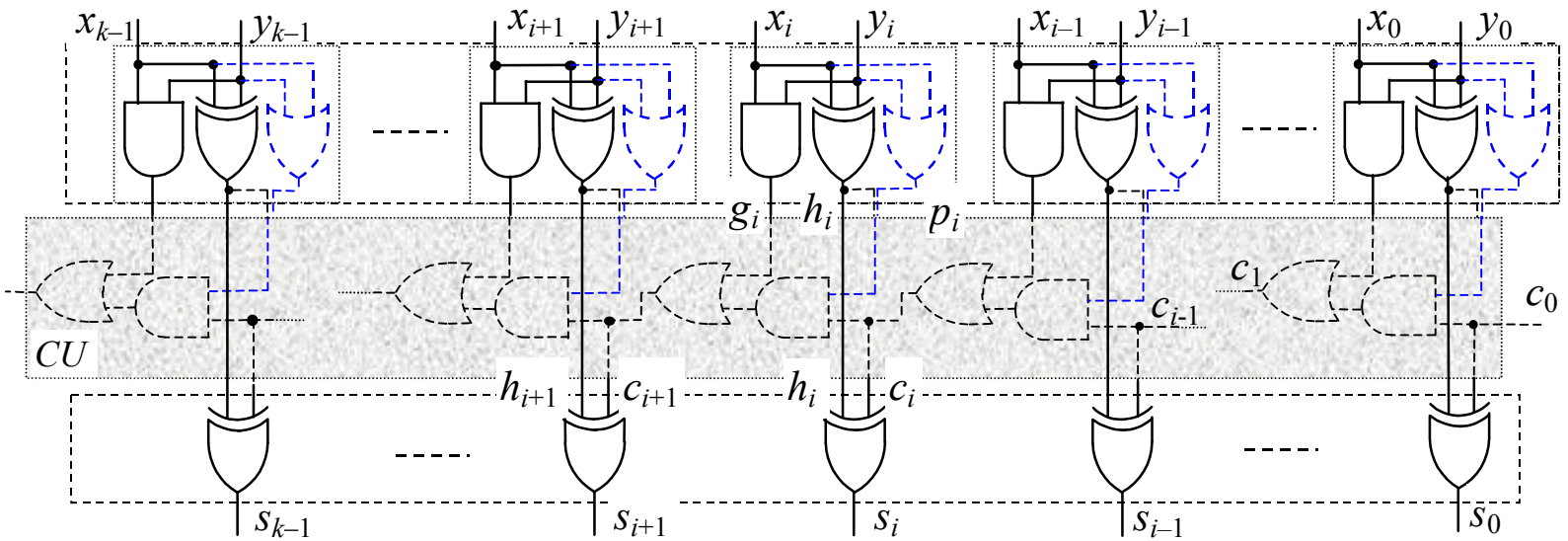




Schemat dodawania/odejmowania w systemach dwójkowych

W podstawowym układzie dodawania/odejmowania można wyróżnić 3 bloki:

- blok wejściowy do równoległego tworzenia funkcji pomocniczych h_i, g_i, p_i(..=h_i)
- blok wytwarzania przeniesień c_i
- blok tworzenia sum s_i



Sumator kaskadowy RCA (ang. *Ripple-Carry Adder*) – sekwencyjne tworzenie przeniesień (w układzie odejmującym zmienia się tylko struktura bloku wejściowego)

Analiza szybkości dodawania i odejmowania

* czas sekwencyjnego dodawania/odejmowania n-pozycyjnego jest rzędu nT
* w systemie dwójkowym: suma lub różnica jest wartością funkcji logicznej $2n$ zmiennych, którą można wytworzyć w układzie o głębokości logicznej $\lceil \log_2 2n \rceil$

Propagacja przeniesienia

- obliczenie sumy/różnicy na pozycji i+1 wymaga użycia przeniesienia z pozycji i:

$$s_i = x_i \oplus y_i \oplus c_i = h_i \oplus c_i$$

- kolejne przeniesienia są powiązane rekurencyjnie:

$$c_{i+1} = g_i + p_i c_i$$

- gwarantowany czas wykonania dodawania/odejmowania zależy od szybkości *wytworzenia przeniesienia* na najwyższej pozycji
- czas T wytworzenia sumy/różnicy – stały od chwili ustalenia przeniesienia

Inne metody

W sieciach logicznych efekt przyśpieszenia można uzyskać także przez:

- *tworzenie i składanie alternatywnych sum pozycyjnych* ($s_i^0 = h_i \oplus 0$, $s_i^1 = h_i \oplus 1$) *oraz grupowych (dla alternatywnych wartości przeniesień wejściowych do grupy pozycji)*
- *wykorzystanie nadmiarowej reprezentacji argumentów (np. kod SD)*

Przyspieszanie dodawania dwuargumentowego

Przyśpieszanie wytwarzania przeniesień

- antycypacja przeniesień (*carry look-ahead adder*, CLA)
  – tworzenie przeniesień dla kilku (zwykle 4) sąsiednich pozycji
- równoległe wytwarzanie przeniesień (*parallel prefix adder*, PPA), albo
  - korekcja sum tymczasowych aktualizowanym przeniesieniem (ELM)
- skracanie ścieżki propagacji przeniesienia (*carry skip adder*, CSKA)

Składanie sum tymczasowych

- sumator z przełączaniem sum częściowych (*carry-select adder*, CSLA)
  – równoległe wytwarzanie alternatywnych blokowych sum częściowych
  - składanie sum warunkowych (*conditional sum adder*, COSA)
    – tworzenie wariantowych sum dla bloków 2^i kolejnych pozycji
- korekcja półsum (*carry-increment adder*, CIA)
  – korekcja sum blokowych przeniesieniami

Składanie sum częściowych jest de facto zrównolegleniem wytwarzania sum wraz z przeniesieniami sterującymi tym składaniem ...

Wytwarzanie i propagacja przeniesień w dodawaniu dwójkowym

Funkcja przeniesienia może mieć jedną z równoważnych form

$$c_{i+1} = x_i y_i + (x_i \oplus y_i)\, c_i = x_i y_i + (x_i + y_i)\, c_i$$

ponieważ $a+b=a\oplus b+ab$ ($OR(a,b)=XOR(a,b)+ab$). Składowymi wyrażenia są:

- funkcja *wytwarzania* (*generowania*) przeniesienia, określająca stan, który wymusza przeniesienie wyjściowe $c_{i+1}=1$, niezależnie od c_i:

$$g_i = x_i y_i,$$

- funkcja *przekazywania* (*propagacji*) przeniesienia ($x_i \neq y_i \Rightarrow c_{i+1} = c_i$)

$$p_i = x_i \oplus y_i \quad \text{lub} \quad p_i = x_i + y_i = \overline{k}_i$$

która jednocześnie określa tzw. *pół-sumę*

$$h_i = x_i \oplus y_i$$

W wyrażeniach określających przeniesienia może też być użyta

- funkcja *wygaszania* (*kasowania*) przeniesienia (wymuszenie $c_{i+1}=0$)

$$k_i = \overline{x_i + y_i}$$

Wytwarzanie i propagacja przeniesień w odejmowaniu dwójkowym

Funkcja *pożyczki* (przeniesienia wstecznego) może mieć jedną z form

$$c_{i+1} = \overline{x}_i y_i + (\overline{x}_i \oplus y_i) c_i = \overline{x}_i y_i + (\overline{x}_i + y_i) c_i$$

ponieważ $a+b=a\oplus b+ab$ ($OR(a,b)=XOR(a,b)+ab$). Składowymi wyrażenia są:

- funkcja *wytwarzania* (*generowania*) pożyczki, określająca stan, który wymusza *pożyczkę* z wyższej pozycji c_{i+1}=1, niezależnie od c_i:

$$g_i = \overline{x}_i y_i,$$

- funkcja *propagacji* (*przekazywania*) pożyczki ($x_i = y_i \Rightarrow c_{i+1} = c_i$):

$$p_i = \overline{x}_i \oplus y_i \quad \text{lub} \quad p_i = \overline{x}_i + y_i = \overline{k}_i$$

która jednocześnie określa tzw. *pół-różnicę*

$$h_i = \overline{x}_i \oplus y_i$$

W wyrażeniach określających pożyczki równoważnie może też być użyta

- funkcja *wygaszania* (*kasowania*) pożyczki (wymuszenie c_{i+1}=0)

$$k_i = \overline{\overline{x}_i + y_i}$$

Powiązania przeniesień*)

Struktura logiczna sieci przeniesień i pożyczek jest jednakowa

$$c_{i+1} = g_i + p_i c_i$$
$$c_{i+2} = g_{i+1} + p_{i+1} g_i + p_{i+1} p_i c_i = (g_{i+1} + p_{i+1} g_i) + (p_{i+1} p_i) c_i$$
$$c_{i+3} = g_{i+2} + p_{i+2}(g_{i+1} + p_{i+1} g_i + p_{i+1} p_i c_i) = [g_{i+2} + p_{i+2}(g_{i+1} + p_{i+1} g_i)] + [p_{i+2}(p_{i+1} p_i)] c_i$$

gdzie $g_i = x_i y_i$, $p_i = x_i \oplus y_i$ w dodawaniu, albo $g_i = \overline{x}_i y_i$, $p_i = \overline{x}_i \oplus y_i$ w odejmowaniu.

Jak widać, każde z tych wyrażeń zawiera dwa składniki powiązane rekurencyjnie:

- opisujący warunki wystarczające do *generowania* (*wymuszenia*) przeniesienia wyjściowego =1 niezależnie od przeniesienia wejściowego,
- określający *propagację* (*przepływ*) przeniesienia z niższej pozycji do wyjścia.

Oznaczając te składniki jako: ($i \geq j$)

generowanie: $G_{i:j} = g_i + p_i g_{i-1} + p_i p_{i-1} g_{i-2} + \ldots + p_i \ldots p_{j+1} g_j$, $G_{i:i} = g_i$

propagacja: $P_{i:j} = p_i p_{i-1} \ldots p_j$, $P_{i:i} = p_i$

otrzymujemy ogólną zależność $c_{i+1} = G_{i:j} + P_{i:j} c_j$

Powiązania funkcji $G_{i:j}$ i $P_{i:j}$ i związki przeniesień można łatwo opisać przy użyciu *podstawowego operatora przeniesień* znanego też jako *operator Brenta-Kunga*.

Podstawowy operator przeniesień

Podstawowy operator przeniesień (ang. *fundamental carry operator*) jest zdefiniowany tak:

$$(f,g)=(x,y)\circ(v,z)=(x+yv,yz)$$

opisuje *rekurencyjne powiązanie kolejnych przeniesień*:

$$(c_{i+1},\ldots)=(g_i,p_i)\circ(c_i,\ldots)$$

Operator ten jest obustronnie **łączny** (ang. *associative*) lecz nie jest przemienny:

$$[(x,y)\circ(q,r)]\circ(a,b)=(x+yq,yr)\circ(a,b)=(x+yq+yra,yrb),$$
$$(x,y)\circ[(q,r)\circ(a,b)]=(x,y)\circ(q+ra,rb)=(x+yq+yra,yrb).$$

Przy jego użyciu powiązanie dowolnych przeniesień można przedstawić jako ($i\geq j$):

$$(c_{i+1},\ldots)=(g_i,p_i)\circ(g_{i-1},p_{i-1})\circ\ldots\circ(g_j,p_j)\circ(c_j,\ldots)=(G_{i:j}+P_{i:j})\circ(c_j,\ldots)$$

Ale każde przeniesienie zależy od c_0, więc (*argument ... jest nieistotny*, może być 0)

$$(c_{i+1},\ldots)=(g_i,p_i)\circ\ldots\circ(g_2,p_2)\circ(g_1,p_1)\circ(g_0,p_0)\circ(c_0,\ldots)=(G_{i:0}+P_{i:0}c_0,\ldots)$$

Rekurencyjnie skojarzone są też funkcje $(G_{i:j},P_{i:j})=(g_i,p_i)\circ(g_{i-1},p_{i-1})\circ\ldots\circ(g_j,p_j)$

$$\begin{aligned}G_{i:j}&=G_{i:k}+P_{i,k}G_{k-1:j}\\P_{i:j}&=P_{i,k}P_{k-1:j}\end{aligned},\quad i\geq k>j$$

Obliczanie funkcji rekurencyjnie skojarzonych – *problem prefiksowy*.

Tworzenie przeniesień

Z uwagi na łączność operatora Brenta-Kunga, poszczególne przeniesienia:

$$(c_{i+1},0) = (g_i,p_i)\circ\ldots\circ(g_3,p_3)\circ(g_2,p_2)\circ(g_1,p_1)\circ(g_0,p_0)\circ(c_0,0) = (G_{i:0}+P_{i:0}c_0,0) =$$
$$= (g_i,p_i)\circ\ldots\circ(g_3,p_3)\circ(g_2,p_2)\circ(g_1,p_1)\circ(g_0+p_0c_0,0) = (G^*_{i:0},0)$$

można wytwarzać wieloma sposobami:

– sekwencyjnie (po kolei) – struktura RC (ang. *Ripple-Carry*):
$(c_1,0) = (g_0,p_0)\circ(c_0,0)$, $(c_2,0) = (g_1,p_1)\circ(c_1,0)$, $(c_3,0) = (g_2,p_2)\circ(c_2,0)$,...

– jednocześnie w grupach po k pozycji – struktura CL (ang. *Carry Lookahead*):

$(c_{i+1},0) = (g_i,p_i)\circ(c_i,0)$,
$(c_{i+2},0) = (g_{i+1},p_{i+1})\circ(g_i,p_i)\circ(c_i,0)$,
$(c_{i+3},0) = (g_{i+2},p_{i+2})\circ(g_{i+1},p_{i+1})\circ(g_i,p_i)\circ(c_i,0)$,
...
$(c_{i+k},0) = (g_{i+k-1},p_{i+k-1})\circ\ldots\circ(g_{i+1},p_{i+1})\circ(g_i,p_i)\circ(c_i,0)$,

– w pełni równolegle – struktura prefiksowa PP (ang. *Parallel Prefix*), np. tak:

$$(c_{i+1},0) = (g_i,p_i)\circ\ldots\circ[(g_5,p_5)\circ(g_4,p_4)]\circ[(g_3,p_3)\circ(g_2,p_2)]\circ[(g_1,p_1)\circ(g_0,p_0)]\circ(c_0,0) =$$
$$= (g_i,p_i)\circ\ldots\circ[(g_5,p_5)\circ(g_4,p_4)]\}\circ\{[(g_3,p_3)\circ(g_2,p_2)]\circ[(g_1,p_1)\circ(g_0,p_0)]\}\circ(c_0,0)$$

Sumator z antycypacją przeniesień (*carry look-ahead adder, CLA*)

Kolekcję przeniesień na kolejne pozycje

$$(c_{i+1},0) = (g_i, p_i) \circ (c_i,0),$$
$$(c_{i+2},0) = (g_{i+1}, p_{i+1}) \circ (g_i, p_i) \circ (c_i,0),$$
$$(c_{i+3},0) = (g_{i+2}, p_{i+2}) \circ (g_{i+1}, p_{i+1}) \circ (g_i, p_i) \circ (c_i,0),...$$

można tworzyć jednocześnie jako funkcje logiczne wielu zmiennych (np. k=4):

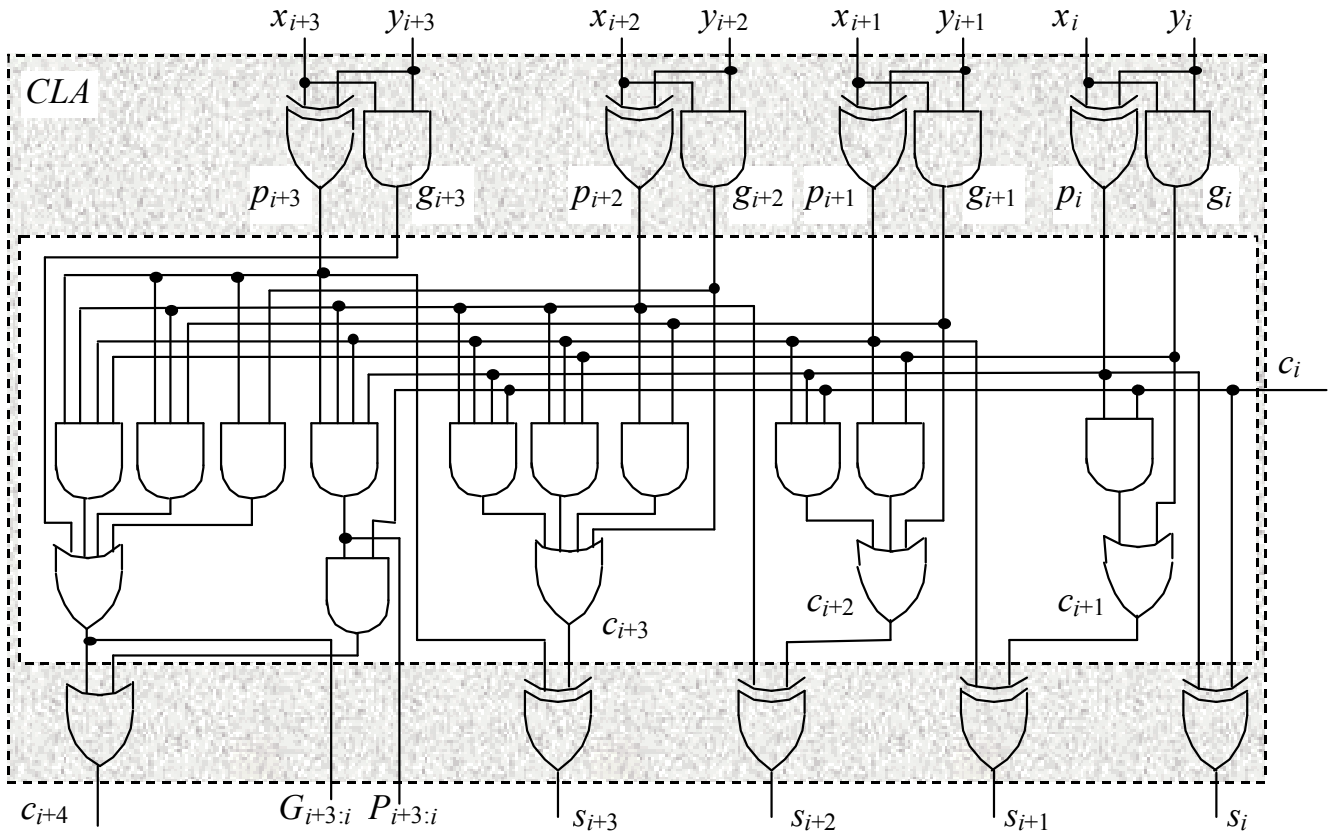
$$c_{i+1} = g_i + p_i c_i$$
$$c_{i+2} = g_{i+1} + p_{i+1} g_i + p_{i+1} p_i c_i$$
$$c_{i+3} = g_{i+2} + p_{i+2} g_{i+1} + p_{i+2} p_{i+1} g_i + p_{i+2} p_{i+1} p_i c_i$$
$$c_{i+4} = g_{i+3} + p_{i+3} g_{i+2} + p_{i+3} p_{i+2} g_{i+1} + p_{i+3} p_{i+2} p_{i+1} g_i + p_{i+3} p_{i+2} p_{i+1} p_i c_i$$

i odpowiednio wyznaczyć kilka kolejnych bitów sumy, ale:

- złożoność funkcji c_{i+r} rośnie z kwadratem zasięgu r,
- ograniczona jest liczba wejść bramki i jej obciążenie prądowe (technologia)

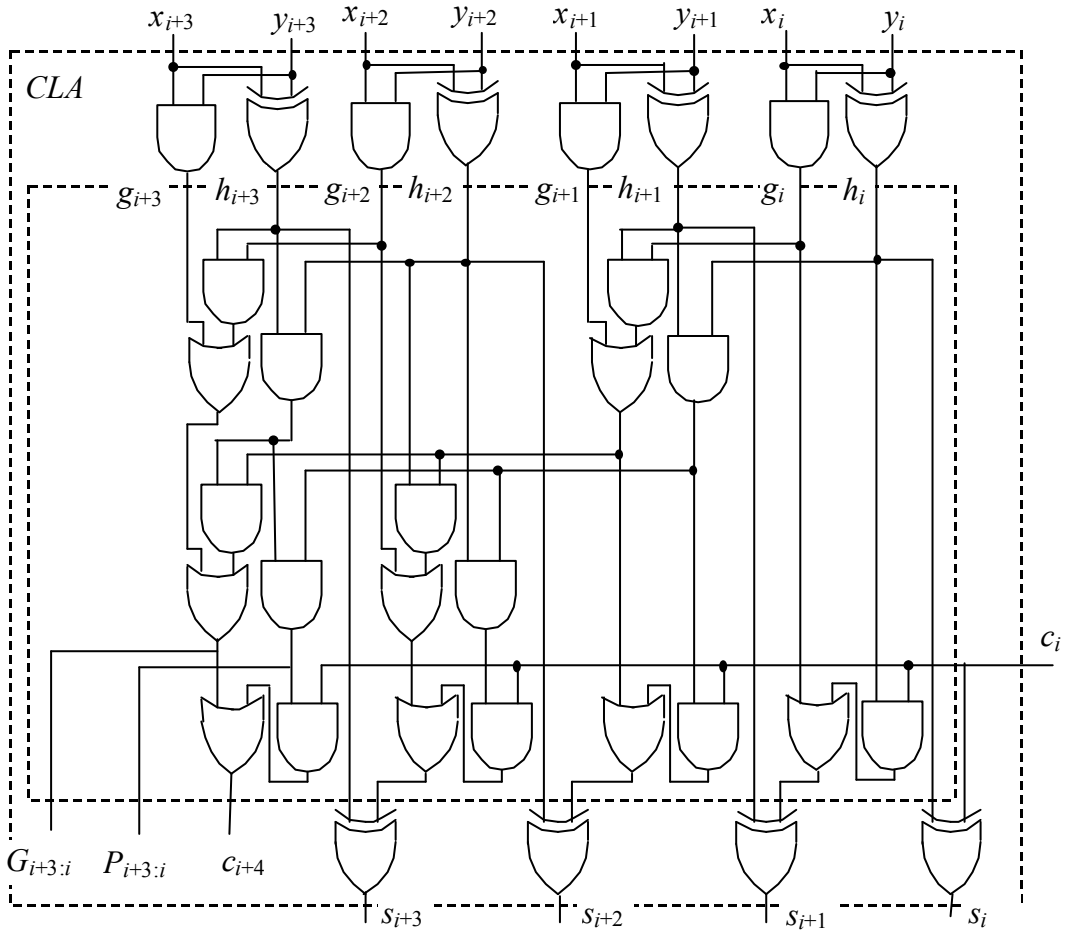
Wyznaczywszy funkcje $(G_{i,i+r-1}, P_{i,i+r-1})$ można za ich pomocą powiązać r-bitowe moduły CLA (typowo r=4) w sumator $n{\cdot}r$-pozycyjny, tworząc blok CLG.

Moduł sumatora z antycypacją przeniesień (CLA)

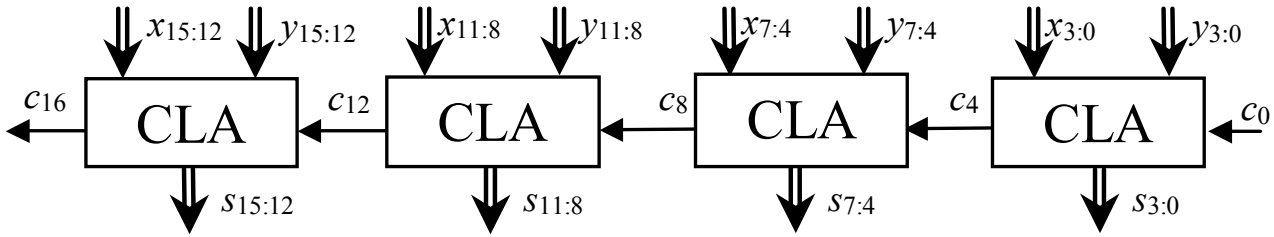


Czterobitowy sumator CLA z sygnałami G,P dla bloku CLG

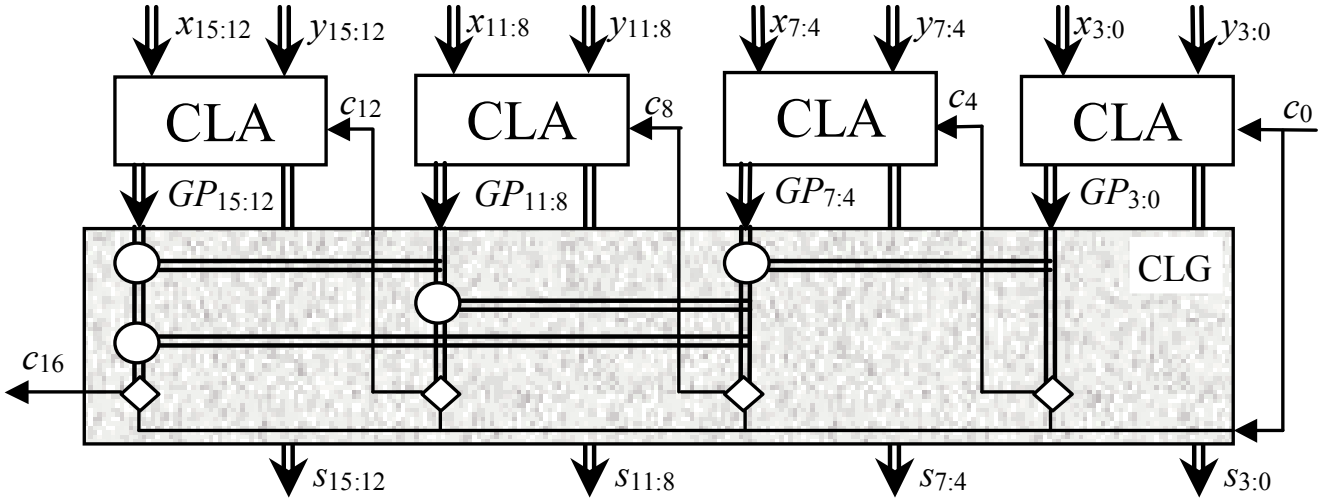
Alternatywny moduł CLA (bramki 2-we)



Wielomodułowe sumatory z antycypacją przeniesień (CLA)

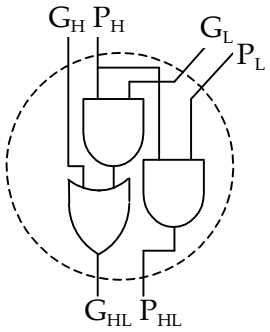


Sumator zbudowany z kaskady bloków CLA



Sumator 16-bitowy CLA z blokiem wytwarzania przeniesień CLG

Struktura logiczna funkcji G i P



$$(G_{HL}, P_{HL}) = (G_H, P_H) \circ (G_L, P_L) = (G_H + P_H G_L,\ P_H P_L)$$

W bloku sumatora/subtraktora, który obejmuje pozycje od i do k (HL= i:k) i zawiera sąsiadujące bloki H=k:s+1 oraz L=s:i ($i \le s \le k$) mamy odpowiednio:

$$G_{k:i} = G_{k:s+1} + P_{k:s+1} G_{s:i},$$
$$P_{k:i} = P_{k:s+1} P_{s:i}$$

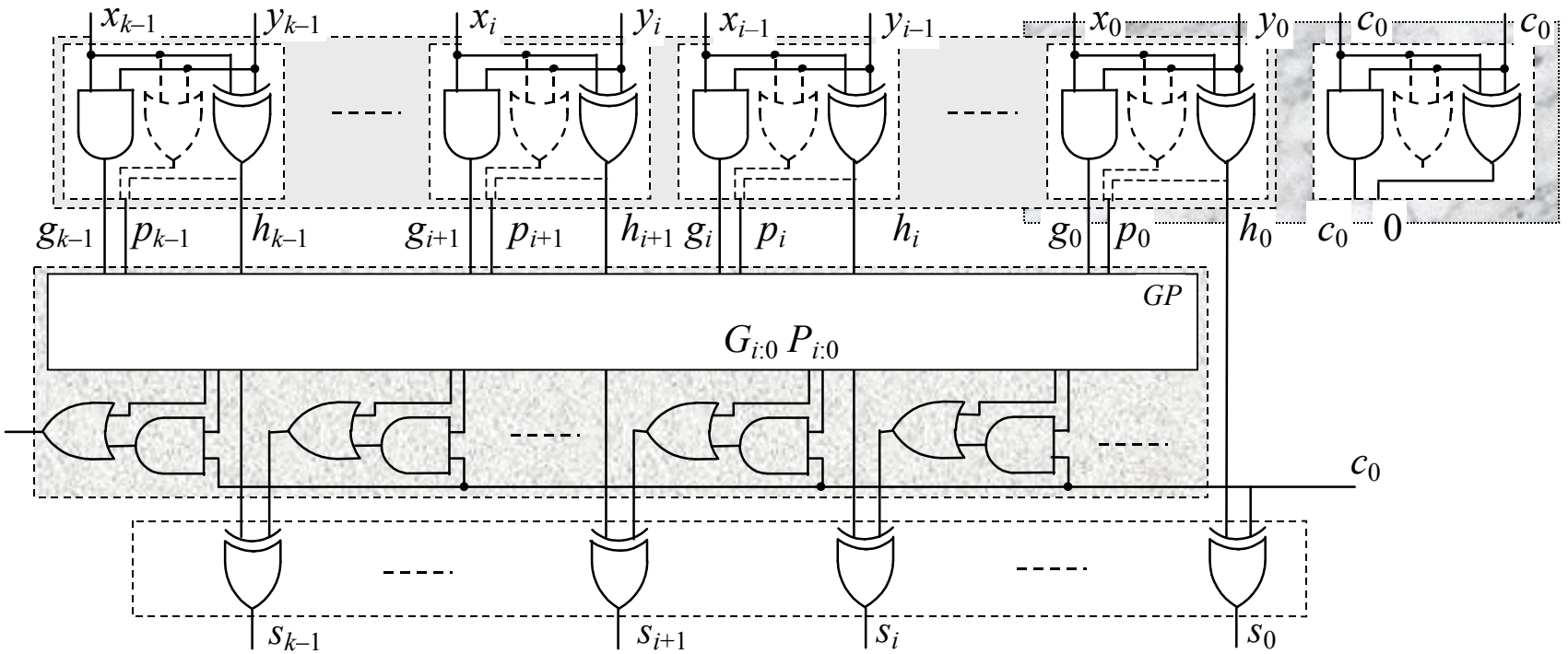
przy tym: $G_{k:k} = g_k = x_k y_k$ i $P_{k:k} = p_k = x_k + y_k$ (lub $P_{k:k} = h_k = x_k \oplus y_k$).

– wartość bitu sumy s_i zależy od półsumy h_i, oraz przeniesienia c_i: $s_i = h_i \oplus c_i$

– przeniesienie zależy od funkcji G oraz P: $c_i = G_{i-1:0} + P_{i-1:0} c_0$

– wytworzenie funkcji $G_{i:0}, P_{i:0}$ można wykonać w sekwencji $\lceil \log_2 n \rceil$ działań

Podstawowa struktura sumatora prefiksowego (PPA)



sumator prefiksowy (ang. *parallel prefix adder,* PPA) – blok GP:

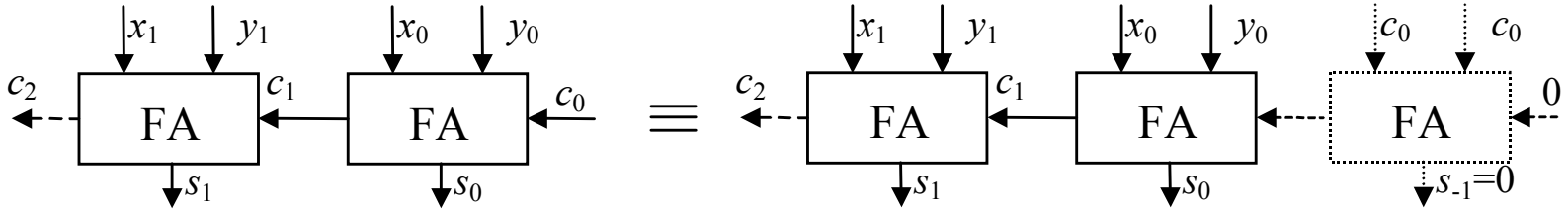
– przeniesienia równoległe $c_i = G_{i-1:0} + P_{i-1:0}c_0$, zamiast sekwencji $c_{i+1} = g_i + p_i c_i$

$$s_i = h_i \oplus c_i = h_i \oplus (G_{i-1:0} + P_{i-1:0}c_0)$$

– jeśli c_0=0, to $c_i = G_{i-1:0}$ i wtedy $s_i = h_i \oplus c_i = h_i \oplus G_{i-1:0}$, $s_0 = h_0$

Sumator prefiksowy rozszerzonej precyzji z redukcją rozgałęzienia c_{IN}

problem: silne rozgałęzienie przeniesienia wejściowego c_{in}, gdy $c_0 \neq 0$, ale:



Uniknięcie rozgałęziania $c_{in}=c_0$ w obliczaniu $c_i = G_{i-1:0} + P_{i-1:0}c_0$ można więc uzyskać traktując c_0 jako przeniesienie z pozycji „–1” , gdy jednocześnie $c_{-1}=0$ i $s_{-1}=0$. Tak będzie, gdy $x_{-1}=y_{-1}=c_0$. Wtedy $g_{-1}=c_0$, $p_{-1}=0$ oraz $s_{-1}=0$ (poprawne rozszerzenie), co jest równoważne zastąpieniu sygnału g_0 przez

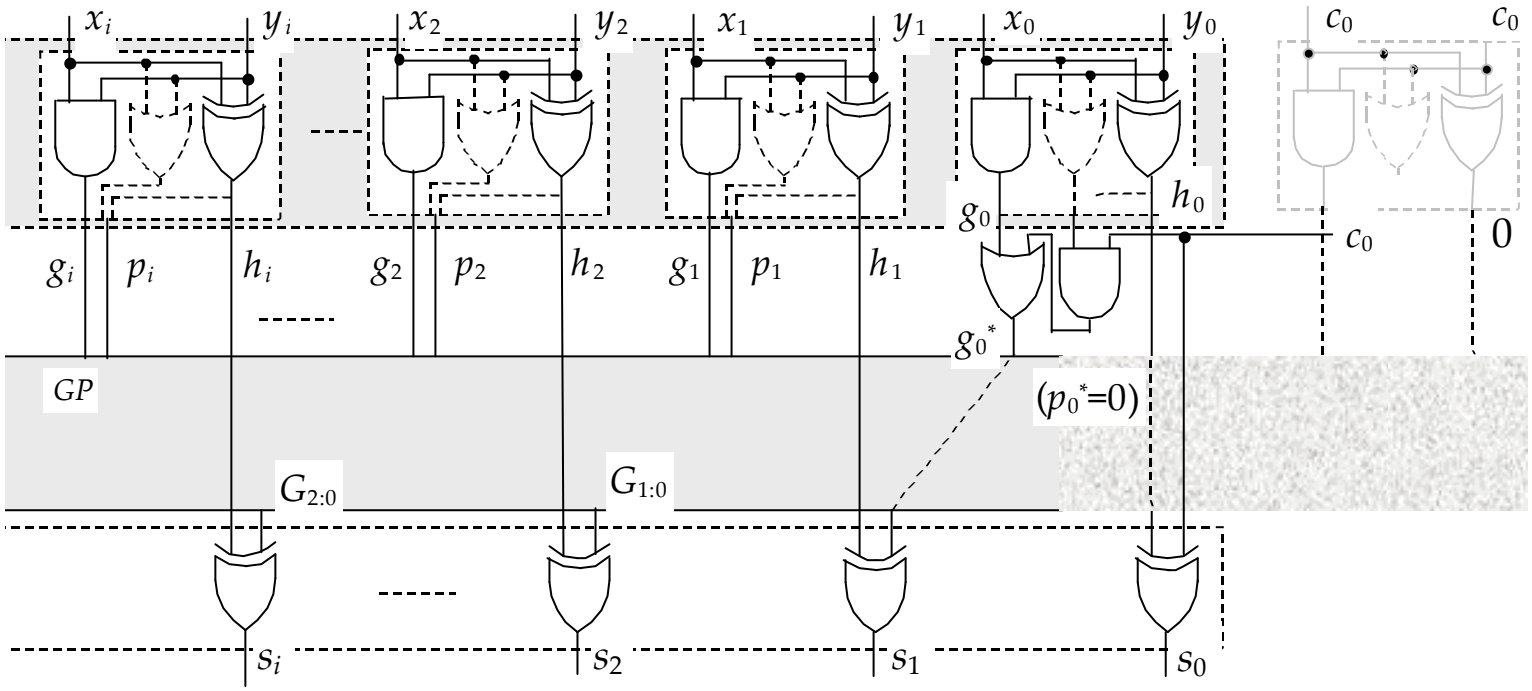
$$g_0^* = G_{0:-1} = g_0 + p_0 g_{-1} = g_0 + p_0 c_0$$

Alternatywny sygnał $p_0^* = P_{0:-1} = p_0 p_{-1} = 0$, bo $p_{-1}=0$, wtedy $P_{i-1:0}^* = P_{i-1:-1} = 0$ a w sumatorze obejmującym pozycje od –1 do n–1 jest ($s_0 = h_0 \oplus c_0$):

$$s_i = h_i \oplus c_i = h_i \oplus G_{i-1:-1} = h_i \oplus (G_{i-1:1} + P_{i-1:1} g_0^*) = h_i \oplus G_{i-1:0}^*.$$

Rozwiązanie to nie wnosi dodatkowego opóźnienia – korekta wejściowa wnosi takie opóźnienie jak obliczenie c_i.

Sumator prefiksowy rozszerzonej precyzji z redukcją rozgałęzienia c_{IN}



Jeśli c_0 jest przetworzone w bloku wstępnym, to $c_i = G_{i-1:0}$. Są dwa schematy:

A) $(c_\#,0) = \ldots \circ \{(g_3,p_3)\circ(g_2,p_2)\} \circ \{(g_1,p_1)\circ[(g_0,p_0)\circ(c_0,0)]\}$

B) $(c_\#,0) = \ldots \circ (g_3,p_3)] \circ [(g_2,p_2)\circ(g_1,p_1)] \circ [(g_0,p_0)\circ(c_0,0)]\}$

Dodatkowa korzyść - uproszczone wykrywanie nadmiaru U2:

$$ov = G_{n-1:0} \oplus G_{n-2:0} = (g_{n-1} + p_{n-1}G_{n-2:0}) \oplus G_{n-2:0} = \overline{g}_{n-1}G_{n-2:0}$$

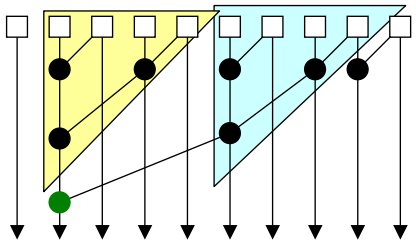
Zasady konstrukcji sieci prefiksowej GP

Węzeł sieci *GP* realizuje **dwie** funkcje (G_{HL} i P_{HL}) **czterech** zmiennych (G_H, P_H, G_L, P_L)

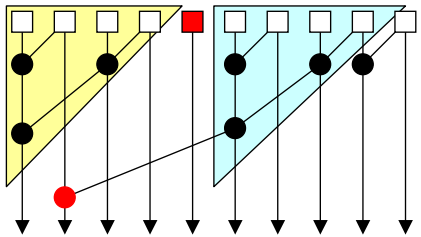
$$(G_{HL}, P_{HL}) = (G_H, P_H) \circ (G_L, P_L) = (G_H + P_H G_L, P_H P_L)$$

Zasady tworzenia struktury *GP* integrującej funkcje G_H, P_H oraz G_L, P_L

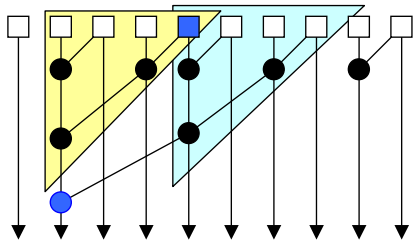
- bloki H i L **powinny być rozdzielnie sąsiadujące**
- bloki H i L **nie mogą być rozdzielone innym blokiem**
- bloki H i L *mogą mieć część wspólną* – funkcje G_{HL} i P_{HL} są **nadmiarowe**



graf optymalny



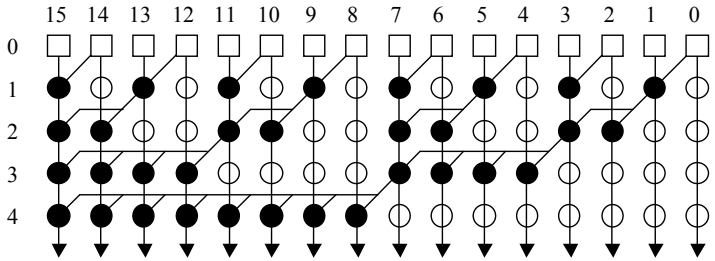
graf błędny



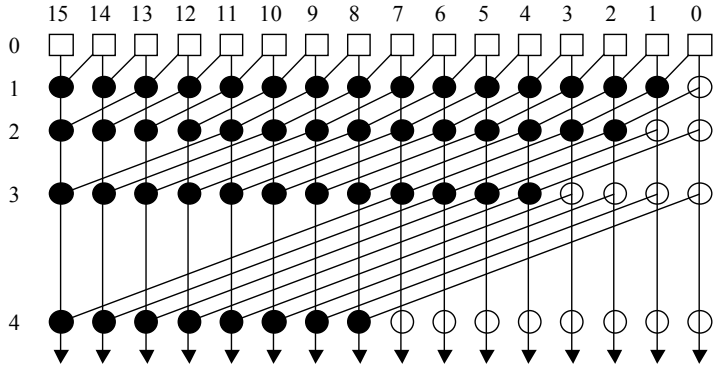
graf poprawny

- □ – wytwarzanie $G_{i:i}=g_i$ i $P_{i:i}=p_i$, ● – operator: $(G_{HL}, P_{HL}) = (G_H, P_H) \circ (G_L, P_L)$
- regularne struktury dla $n=2^k$ wejść (pozycji),
- w innych przypadkach przyjąć $k=\text{int}(1+\log_2 n)$ i usunąć zbędne gałęzie (sieć integrującą 2^{k-1} pozycji połączyć siecią integrującą pozostałe wejścia)

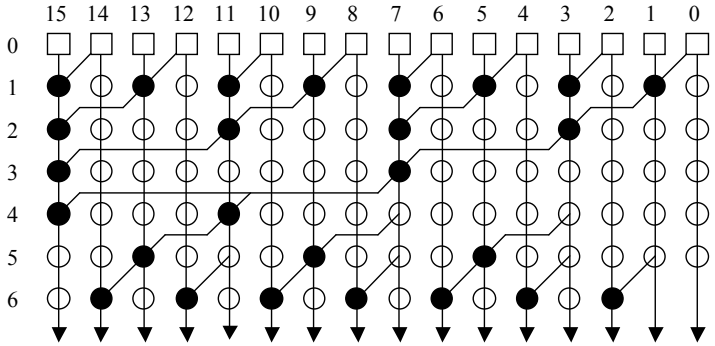
Grafy sieci równoległego generowania i propagacji przeniesienia (PPA)



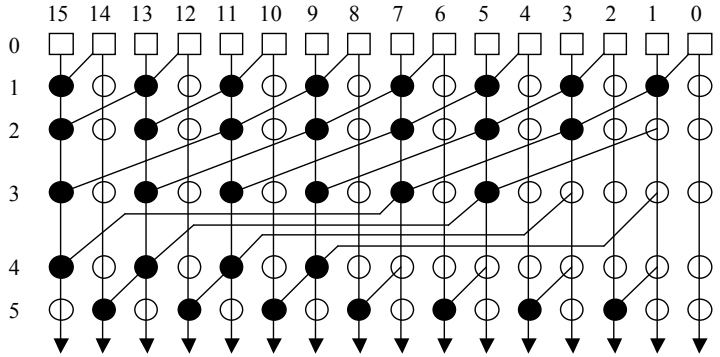
Graf prefixowy (Sklansky / Ladner-Fischer) (16 b)



Graf prefixowy (Kogge & Stone) (16 b)



Graf prefixowy (Brent–Kung) (16 b)



Graf prefixowy – (Han & Carlson) (16 b)

□ – wytwarzanie funkcji $G_{i:i}=g_i$ oraz $P_{i:i}=p_i$ ○ – przekazywanie G, P bez zmiany

● – operator prefiksowy $(G_{HL},P_{HL})=(G_H,P_H)\circ(G_L,P_L)$

Charakterystyki grafów prefiksowych

Ladner-Fischer – $\log_2 n$ poziomów logicznych, minimum elementów *GP* nierównomierne obciążenia (*Sklansky*)

Kogge & Stone – $\log_2 n$ poziomów logicznych, więcej elementów *GP*, rozłożona obciążalność wyjść

Brent-Kung – $>\log_2 n$ poziomów logicznych, mniej elementów *GP*, stała obciążalność wyjść

Han & Carlson – $>\log_2 n$ poziomów logicznych, najmniej elementów *GP*, najmniejsza obciążalność wyjść

Parametry sieci GP jako elementy PPA

Typ struktury	liczba ogniw GP	l. poziomów	obciążenie	przełączenia
RCA	$^2/_3\, n$	$n-1$	2	$n/2$
Ladner-Fischer	$\frac{1}{2}\, n\log_2 n$	$\log_2 n$	$n/2$	$\frac{1}{4}\, n\log_2 n$
Brent-Kung	$2n-n\log_2 n-2$	$2\log_2 n-2$	$\log_2 n+1$	$\sim {}^3/_8\, n\log_2 n$
Kogge & Stone	$n\log_2 n-n+1$	$\log_2 n$	2	$\frac{1}{2}\, n\log_2 n$
Han & Carlson	$\frac{1}{2}\, n\log_2 n$	$\log_2 n+1$	2	$\frac{1}{4}\, n\log_2 n$

Sumator ELM - koncepcja*

Obliczona wartość początkowa sumy h_i=$s_{i:i}$ na może ulec zmianie, jeśli c_i=1.
Niech $s_{i:r}$ oznacza *tymczasową* sumę na pozycji i z uwzględnieniem wszystkich wcześniejszych wejść $x_\#$, $y_\#$, począwszy od wejścia x_r, y_r. Mamy:

$$s_{i:r} = h_i \oplus G_{i-1:r},$$
$$s_{i:0} = h_i \oplus (G_{i-1:0} + H_{i-1:0} c_0)$$

Z rekurencyjnego powiązania funkcji $G_{i:j}$ wynika dalej, że (i>r>j):

$$s_{i:j} = h_i \oplus G_{i-1:j} = h_i \oplus G_{i-1:r} \oplus G_{i-1:r} \oplus (G_{i-1:r} + H_{i-1:r} G_{r-1:j}) =$$
$$= s_{i:r} \oplus (G_{i-1:r} \oplus (G_{i-1:r} + H_{i-1:r} G_{r-1:j})) = s_{i:r} \oplus \overline{G}_{i-1:r} H_{i-1:r} G_{r-1:j},$$

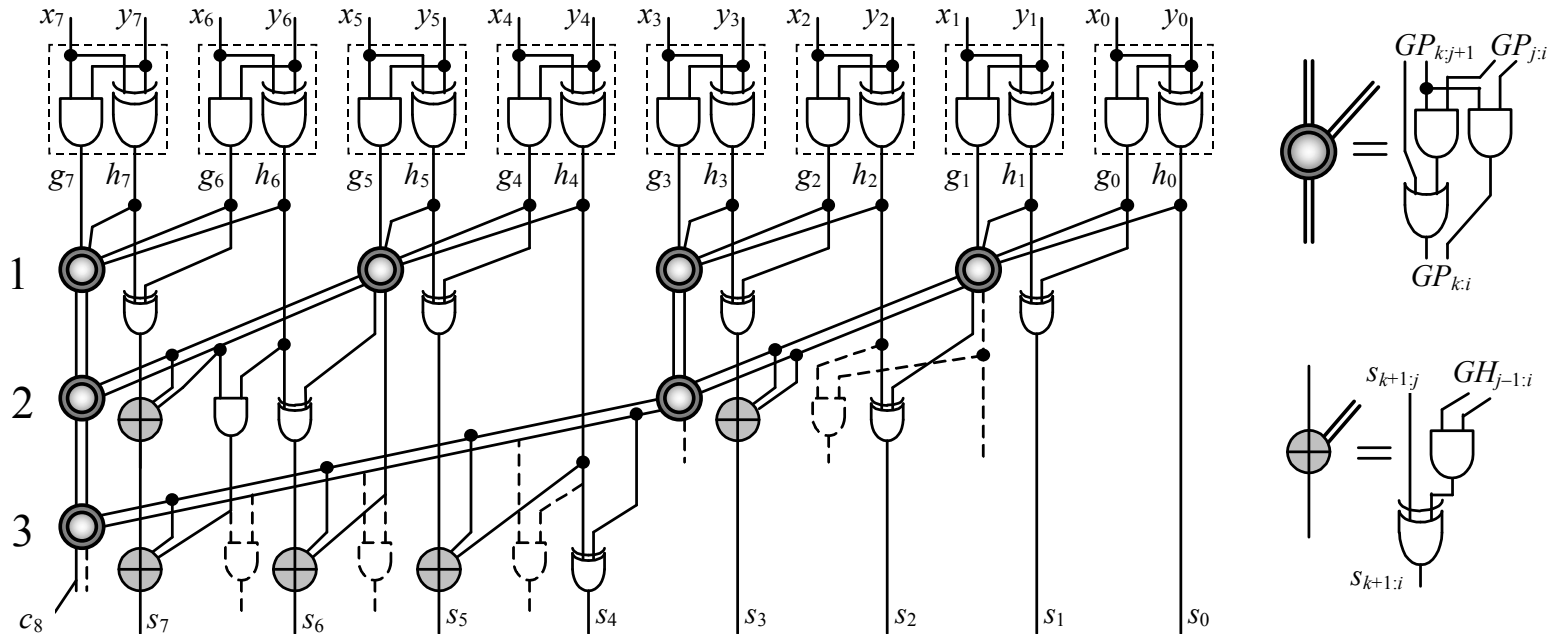
Metodą indukcji można pokazać, że $\overline{G}_{i:j} H_{i:j} = h_i \overline{G}_{i-1:j} H_{i-1:j} = H_{i:j}$, skąd mamy:

$$s_{i:r} = h_i \oplus G_{i-1:r},$$
$$s_{i:j} = s_{i:r} \oplus H_{i-1:r} G_{r-1:j}.$$

przy tym bitami końcowej sumy są s_i=$s_{i:0}$.
Ponieważ powyższe funkcje są niezależne, więc możliwe jest wytworzenie sumy końcowej w strukturze zawierającej $\log_2 n$ poziomów.
Jeśli $c_0 \neq 0$ należy, jak w sumatorze PPA, przyjąć $g_0^* = g_0 + h_0 c_0$ oraz $h_0^* = 0$.

Sumator ELM – korekcja sum tymczasowych*

Z podanych zależności wynika zasada konstrukcji sumatora, podobna jak PPA



Schemat sumatora ELM (struktura Ladnera-Fischera) przy c_0=0.

Powiązania sum można także zrealizować w strukturze Kogge'a-Stone'a

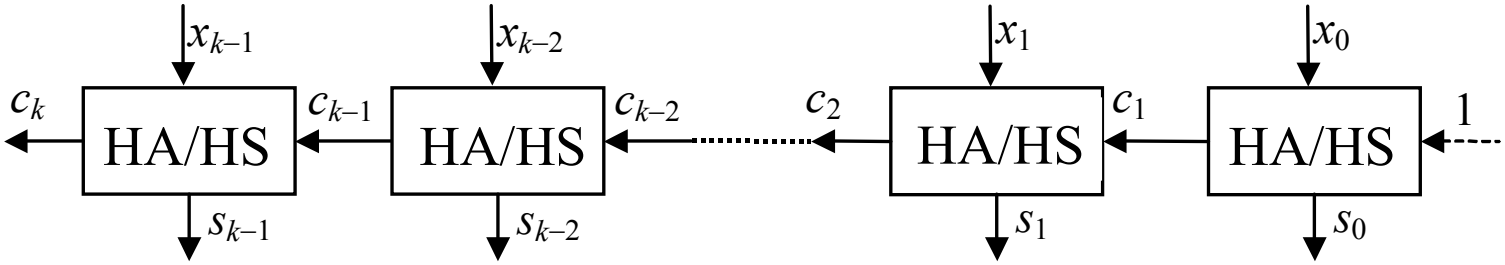
Inkrementer i dekrementer

wykonuje działanie $X+1$ (inkrementer) albo $X-1$ (dekrementer)

$\rightarrow$ wystarczy łańcuch półsumatorów (HA) lub półsubtraktorów (HS)

półsumator (*half adder*, HA) – realizuje funkcje $s_i = x_i \oplus c_i,\ c_{i+1} = x_i c_i$

półsubtraktor (*half subtracter*, HS) – realizuje funkcje $s_i = x_i \oplus c_i,\ c_{i+1} = \overline{x}_i c_i$



Układy inkrementera i dekrementera są *komplementarne*, bo $X-1 = \overline{\overline{X}+1}$

układ zliczający – inkrementer/dekrementer ze sprzężeniem $x_i^{(t+1)} = s_i^{(t)}$

i zapamiętywaniem stanu $S(t) = \{s_{k-1}^{(t)}, s_{k-2}^{(t)}, \ldots, s_1^{(t)}, s_0^{(t)}\}$

Inkrementer i dekrementer prefiksowy

Ponieważ układy inkrementera i dekrementera są *komplementarne*, bo $X-1=\overline{\overline{X}+1}$, wystarczy zbudować jeden z nich. Przeanalizujemy budowę inkrementera.

Założenie: Jedynkę traktujemy jako $c_0=1$, więc argumentami są X, Y=0 i $c_0=1$.

Blok wejściowy:

Na każdej pozycji jest: $g_i=G_{i:i}=x_i\wedge 0=0$, $p_i=P_{i:i}=x_i\oplus 0=x_i=h_i$. Wobec tego:

$$(c_{i+1},0)=(0,x_i)\circ\ldots\circ(0,x_3)\circ(0,x_2)\circ(0,x_1)\circ(0,x_0)\circ(1,0)$$

Stąd wynika, że $c_i=P_{i-1:0}=x_{i-1}\ldots x_1x_0$ oraz $s_i=x_i\oplus P_{i-1:0}$

A zatem:

1. Blok wejściowy jest zbędny ($g_i=0$, $p_i=h_i=x_i$)
2. Węzłem sieci prefiksowej jest więc bramka iloczynu logicznego $P_{HL}=P_HP_L$
3. Blok sumy zawiera 2-wejściowe bramki XOR realizujące $s_i=x_i\oplus P_{i-1:0}$

Dekrementer powstaje przez zanegowanie wszystkich bitów argumentu oraz wszystkich bitów sumy.

SUMATORY WARUNKOWE I INNE

Sumy warunkowe – koncepcja (Sklansky)

L	$x_i + y_i$	1+0	0+0	1+1	1+0	0+1	1+1	1+0	0+1
0	c^0_{i+1}	0	*0*	1	*0*	0	*1*	0	*0*
	$s^0_{i:i}$	1	0	0	1	1	0	1	**1**
	c^1_{i+1}	1	*0*	1	*1*	1	*1*	1	—
	$s^1_{i:i}$	0	1	1	0	0	1	0	—
1	c^0_{2i+2}	0		*1*		*1*		*0*	
	$s^0_{2i+1:2i}$	1	0	0	1	0	0	**1**	**1**
	c^1_{2i+2}	0		*1*		*1*		—	—
	$s^1_{2i+1:2i}$	1	1	1	0	0	1	—	—
2	c^0_{4i+4}	0				*1*			
	$s^0_{4i+3:4i}$	1	1	0	1	**0**	**0**	**1**	**1**
	c^1_{4i+4}	0				—	—	—	—
	$s^1_{4i+3:4i}$	1	1	1	0	—	—	—	—
	$\mathbf{s}^3_{7:0}$	**1**	**1**	**1**	**0**	**0**	**0**	**1**	**1**

Sumator sum warunkowych (*conditional sum adder, COSA*)*

Tworzenie alternatywnych sum jedno-, dwu-, cztero-, ośmio-, ...-bitowych

Poziom 0 – sumy i przeniesienia warunkowe dla osobnych bitów (i=0,1,...)

$$x_i + y_i + 0 = 2c^0_{i:i} + s^0_{i:i} \quad \text{oraz} \quad x_i + y_i + 1 = 2c^1_{i:i} + s^1_{i:i}$$

$$\mathbf{s}_{i:i} = \{s^0_{i:i}, s^1_{i:i}\} = \{x_i \oplus y_i, x_i \equiv y_i\}$$

$$\mathbf{c}_{i+1} = \{c^0_{i+1}, c^1_{i+1}\} = \{x_i y_i, x_i + y_i\}$$

Poziom p (|| – złożenie wektorów)

– warunkowe sumy $\mathbf{s}^{\alpha}_{2ri+2r-1,2ri}$ i przeniesienia $c^{\alpha}_{2r(i+1)}$ grup r=2^p bitów,

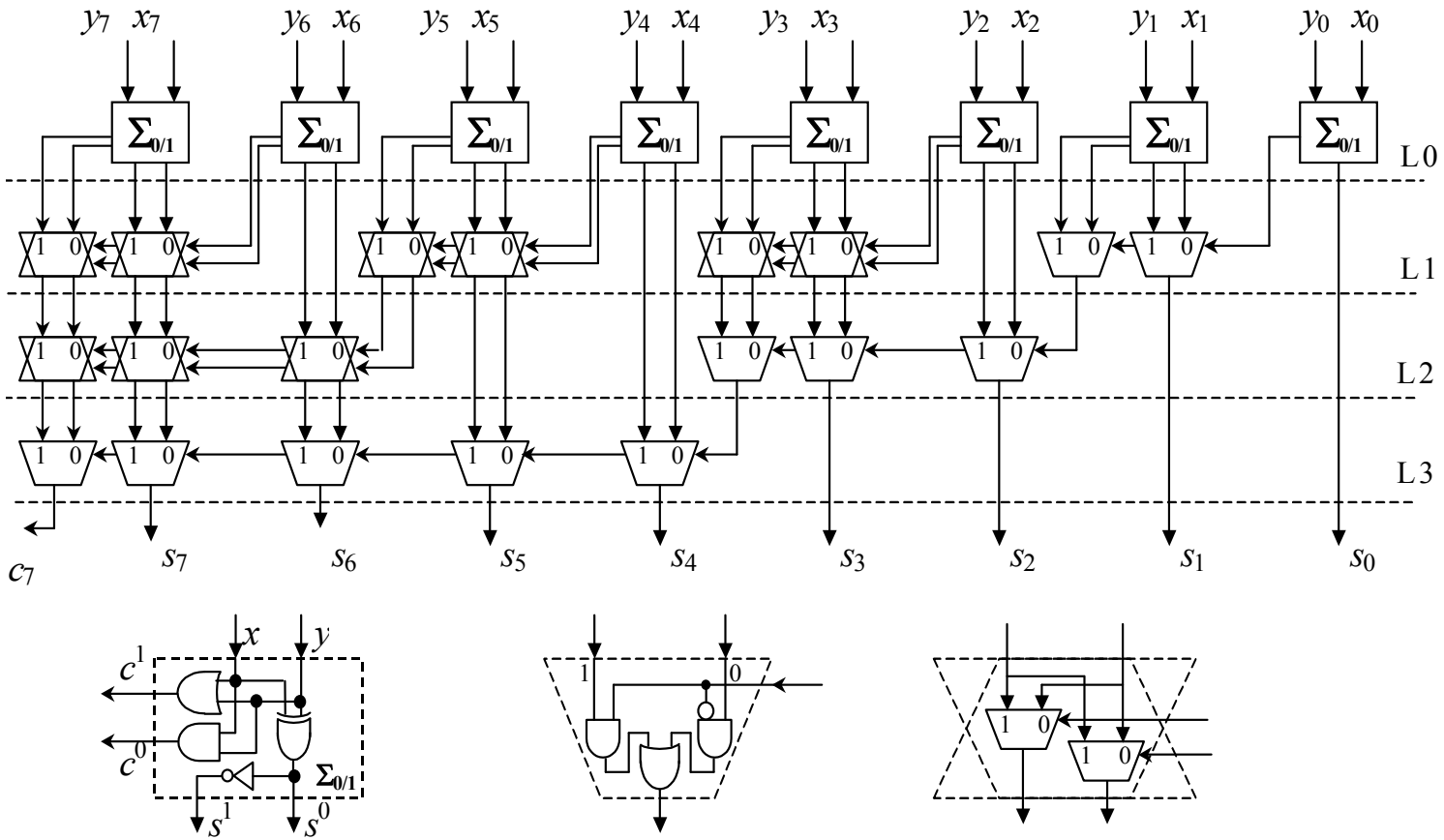
– dla i=0,1,...,$\lceil n \cdot 2^{-p} \rceil - 1$

$$\mathbf{s}^{\alpha}_{2ri+2r-1,2ri} = [c^{\alpha}_{2ri+r}\, \mathbf{s}^1_{2ri+2r-1,2ri+r} + (1 - c^{\alpha}_{2ri+r})\, \mathbf{s}^1_{2ri+2r-1,2ri+r}] \,\|\, \mathbf{s}^{\alpha}_{2ri+r-1,2ri},$$

$$c^{\alpha}_{2r(i+1)} = c^{\alpha}_{2ri+r}\, c^1_{2ri+2r} + (1 - c^{\alpha}_{2ri+r})\, c^0_{2ri+2r}$$

Końcowy wynik sumowania powstaje na poziomie k=$\lceil \log_2 n \rceil$ (r=2^k).

Schemat sumatora sum warunkowych



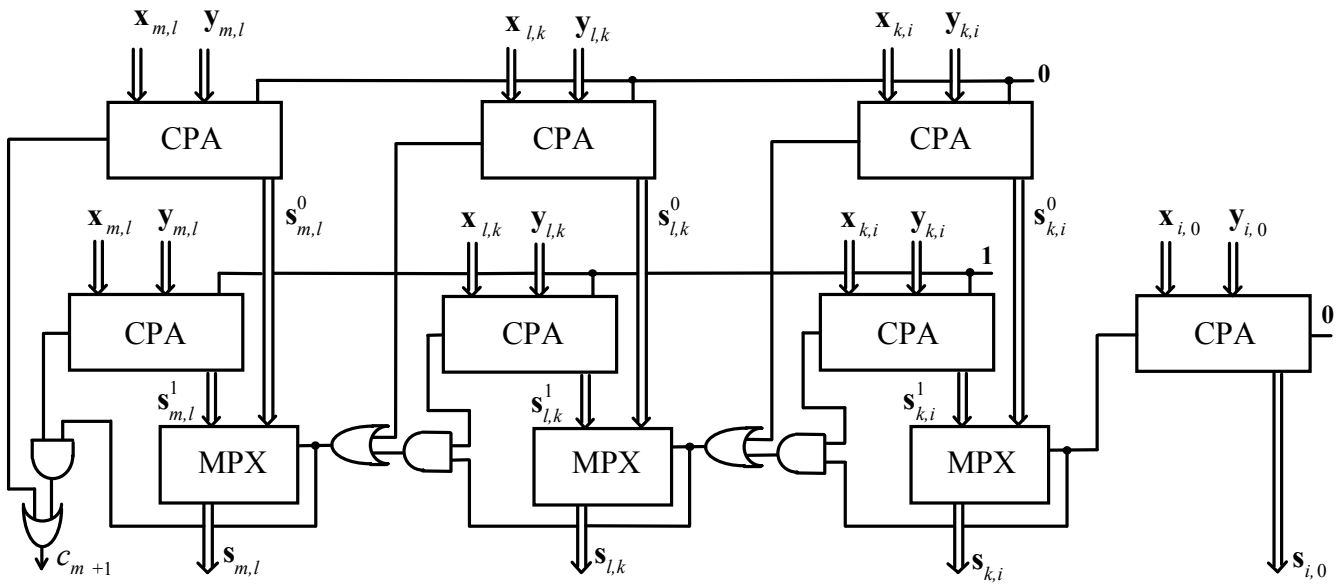
Ośmiobitowy sumator sum warunkowych

$$T = 2\lceil \log_2 2n \rceil,\ A = \tfrac{1}{2}(n\log_2 n + 2n\log_2 n) = 3n\log_2 n$$

Sumator sterowany przeniesieniem (CSLA)

Sumator multipleksowany sterowany przeniesieniem (*carry-select adder*)

składanie k_i-pozycyjnych alternatywnych sum zależnie od przeniesienia tworzonego w szeregowym łańcuchu propagacji



Schemat logiczny sumatora multipleksowanego sterowanego przeniesieniem

Sumy blokowe obliczane jednocześnie $\Rightarrow$ wyższe bity$\rightarrow$większe bloki

Opóźnienie – > $2\sqrt{2n}$ (optymalna liczba bloków – około $\sqrt{2n}$)

Szybkość działania i złożoność sumatorów

Charakterystyki AT

- sumator pełny 1-bitowy FA – $A=7$, $T=2+2 \rightarrow AT=28$
  – 2×XOR, 1×OR, 2×AND $\rightarrow$ opóźnienie przeniesienia 2, sumy 2+2

- sumator RCA – $A=7n$, $T=2n \rightarrow AT=14n^2$
  – n×FA $\rightarrow$ opóźnienie przeniesienia $2n$
- sumator kaskadowy CLA – $A \approx 7n$, $T \approx 4\log n \rightarrow AT \approx 56n\log n$
  – n×FA $\rightarrow \log n$ bloków, opóźnienie przeniesienia $2 \cdot 2\log n$
- sumator PPA – $A \approx 5n+3n\log\sqrt{n}$, $T \approx 3+2\log n \rightarrow AT \approx 3n\log^2 n+14n\log n$
  – $\log n$ poziomów GP, opóźnienie przeniesienia $2\log n$
- sumator COSA – $A=3n\log n$, $T=2+2\log n \rightarrow AT \approx 6n\log^2 n$
  – 2×RCA, $\log n$ poziomów MPX, opóźnienie przeniesienia $2 \cdot \log n$
- sumator CSKA – $A \approx 8n$, $T \approx 2 \cdot 2\sqrt{n} \rightarrow AT \approx 32n\sqrt{n}$
  – n×FA+$2\sqrt{n}$×MPX, $2\sqrt{n}$ bloków $\rightarrow$ opóźnienie przeniesienia $2 \cdot 2\sqrt{n}$
- sumator CSLA – $A \approx 2 \cdot 7n$, $T \approx 2\sqrt{2n} \rightarrow AT \approx 39n\sqrt{n}$
  – 2×RCA, $\sqrt{2n}$ bloków, opóźnienie przeniesienia $2 \cdot \sqrt{2n}$

Moduły sumatorów i subtraktorów i ich połączenia (1)

Sumator/subtraktor – układ realizujący dodawanie/odejmowanie na ustalonej liczbie pozycji, wtedy c_0=0.

Moduł sumatora/subtraktora – element konstrukcyjny realizujący algorytm dodawania/odejmowania pozycyjnego z użyciem przeniesienia/pożyczki.

Podstawowy moduł sumatora/subtraktora dwójkowego – jednopozycyjny sumator/subtraktor dwójkowy (FA/FS) (wejście odjemnej oznaczone):

sumator:

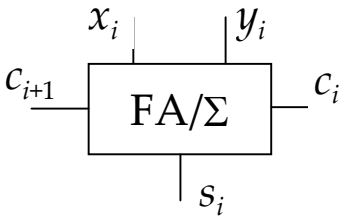
$$s_i = x_i \oplus y_i \oplus c_i = h_i \oplus c_i$$

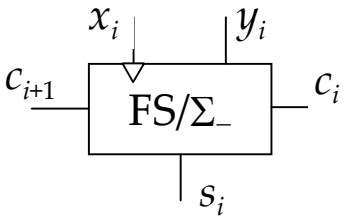
$$c_{i+1} = x_i y_i + (x_i \oplus y_i)\, c_i = x_i y_i + (x_i + y_i)\, c_i = g_i + p_i c_i$$

subtraktor:

$$\bar{s}_i = \bar{x}_i \oplus y_i \oplus c_i = h_i \oplus c_i$$

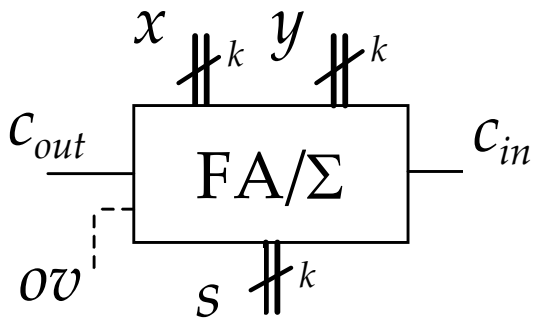
$$c_{i+1} = \bar{x}_i y_i + (\bar{x}_i \oplus y_i)\, c_i = \bar{x}_i y_i + (\bar{x}_i + y_i)\, c_i = g_i + p_i c_i$$

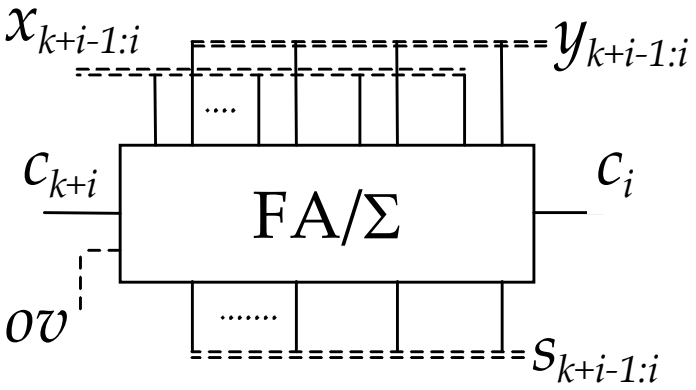


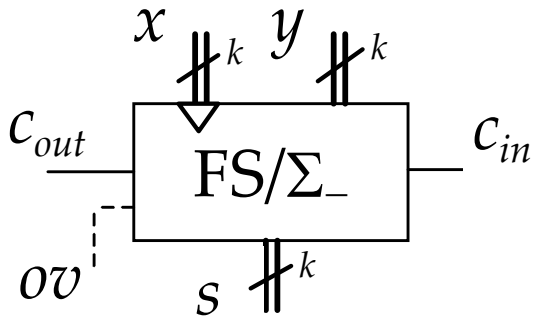


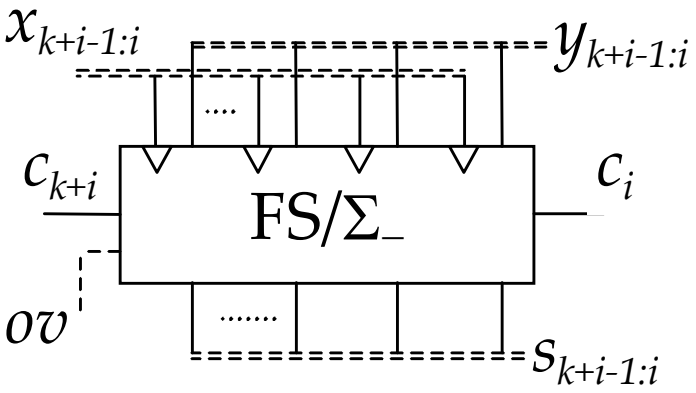
Moduły sumatorów i subtraktorów

Moduł sumatora/subtraktora *k-pozycyjnego* – element konstrukcyjny realizujący algorytm dodawania/odejmowania *jednopozycyjnego* w podstawie β^k (2^k)









Połączenia modułów sumatorów i subtraktorów

Podstawowym objawem przekroczenia zakresu w kodzie uzupełnieniowym jest niezgodność pozycji lewostronnego rozszerzenia sumy/różnicy z wartością obliczoną na najwyższej pozycji dodawania/odejmowania.
Równoważnym objawem jest niezgodność przeniesień na i z najwyższej pozycji.

Moduł dwójkowego **kodu uzupełnieniowego** (U2) **wytwarza** sygnał niezgodności przeniesień na i z najwyższej pozycji $ov = c_{k+i} \oplus c_{k+i-1}$,
moduł dla dwójkowego **kodu naturalnego** (NB) **nie ma** takiego wyjścia.

Moduł przeznaczony do użycia w szybkich układach może mieć również wyprowadzone sygnały modułowej generacji i propagacji przeniesień (np. CLA)