専門基礎科目 - 必修科目

GB10804　論理回路

HPCS-FPGA Lab.
FPGA expert team in High-Performance Computing System Laboratory

システム情報系情報工学域　山口佳樹

専門基礎科目 – 必修科目（情報科学類）
基礎科目 – 関連科目（情報メディア創成学類）
基礎科目 – 関連科目（工学システム学類）

いろいろな組み合わせ回路について考える

- 授業で学んだ算術演算（加算器）から考えてみる。

復習問題

①半加算器を書いてみましょう。

②半加算器を１ブロックとし，それを繋げて全加算器を書いてみましょう。

③全加算器を１ブロックとレジスタを用い，n ビット加算器を書いてみましょう。

ビット直列加算器（BSA）

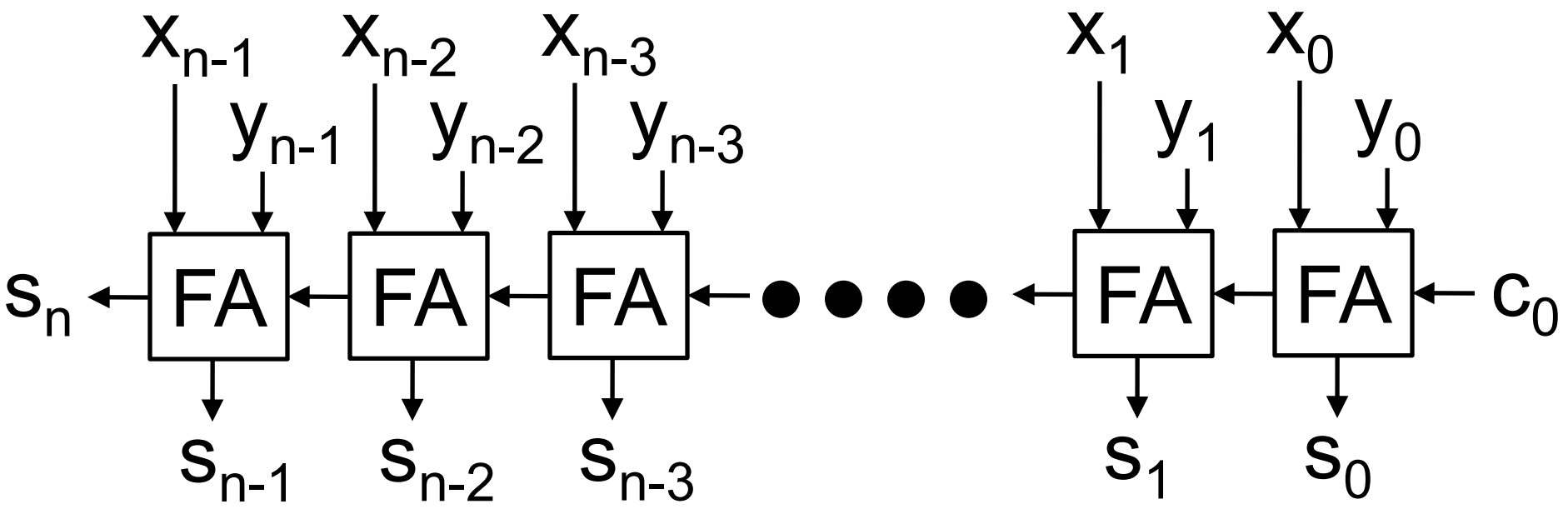
1. 英名： Bit Serial Adder

2. 全加算器(FA)を１個利用

3. 計算時間 O(n), 回路量 O(1)

BSA は直列だから遅い。
⇒ 並列化して早くしよう ⇒ RCA の登場。

順次桁上げ加算器（RCA）



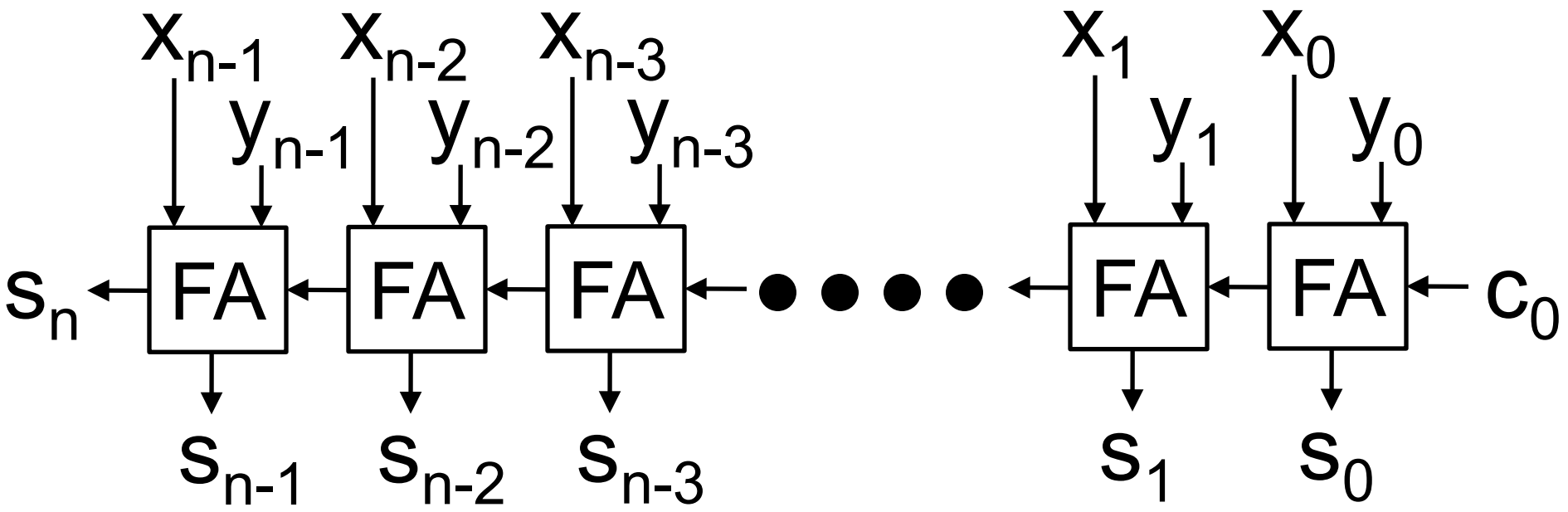
利点：ハードウェア量が少ない
規則正しい1次元構造

順次桁上げ加算器（RCA）

1. 英名： Ripple Carry Adder
2. ビット直列加算器を並列に展開
3. 計算時間，回路量ともに O(n)

この回路では，
　最悪の計算時間＝通常の計算時間
となり非効率 ⇒ 桁上げの有無を判別！

桁上げ完了検出加算器(CCA)



$$c_{i+1}^{1} = x_i \cdot y_i \vee (x_i \oplus y_i) \cdot c_i^{1}$$

$$c_{i+1}^{0} = \overline{x_i} \cdot \overline{y_i} \vee (x_i \oplus y_i) \cdot c_i^{0}$$

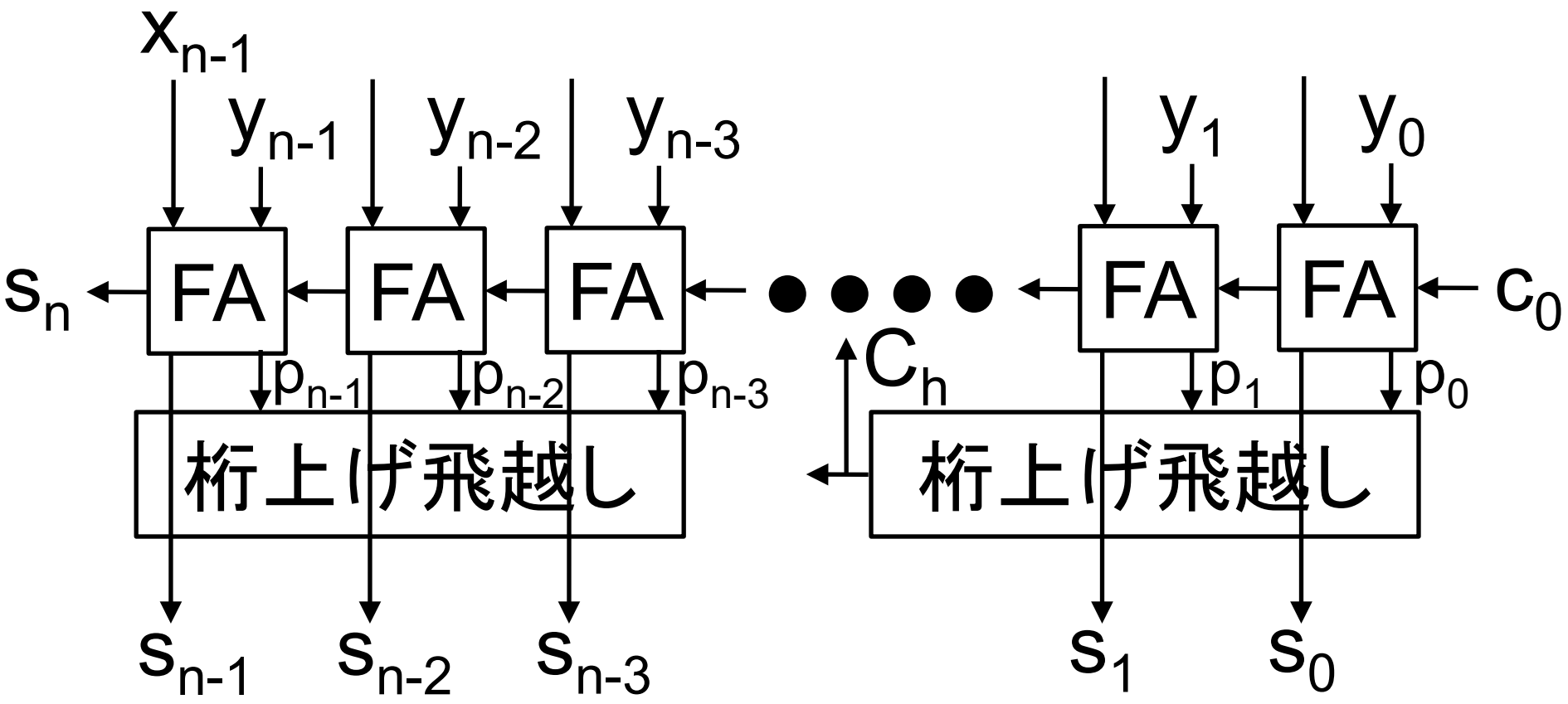
$$cc = (c_1^{1} \vee c_1^{0}) \cdot (c_2^{1} \vee c_2^{0}) \cdots (c_n^{1} \vee c_n^{0})$$

桁上げ完了検出加算器(CCA)

1. 英名： Carry Completion Adder
2. RCAに桁上げ完了検出回路を付加
3. 計算時間，回路量ともに O(n)

検出回路は最悪の計算時間を改善しないため，これを改善する必要があります。
⇒ 桁上げ飛び越し加算器（CBA）

桁上げ飛越し加算器（CBA）



$$P_h = p_j p_{j-1} \cdots p_{j+k-1}$$

$$C_{h+1} = c_{j+k} \vee P_h C_h$$

但し, $p_i = x_i \oplus y_i$

桁上げ飛越し加算器（CBA）

1. 英名： Carry Bypass (or Skip) Adder
2. RCAに桁上げ飛越し回路を付加
3. 計算時間O(√n)～O(n)，回路量 O(n)

ブロックの大きさを√n に比例する一定値にすると，ブロックの個数も√n に比例し，計算時間も√n に比例するようになる.

自由課題（レポート）

16bit の桁上げ飛越し加算器を作成します.
このとき，桁上げ飛越し回路のブロックを

- 4-4-4-4 と均等にした場合
- 1-2-3-4-3-2-1と可変にした場合

とでどちらが高速になるでしょう.
その理由も考えなさい.

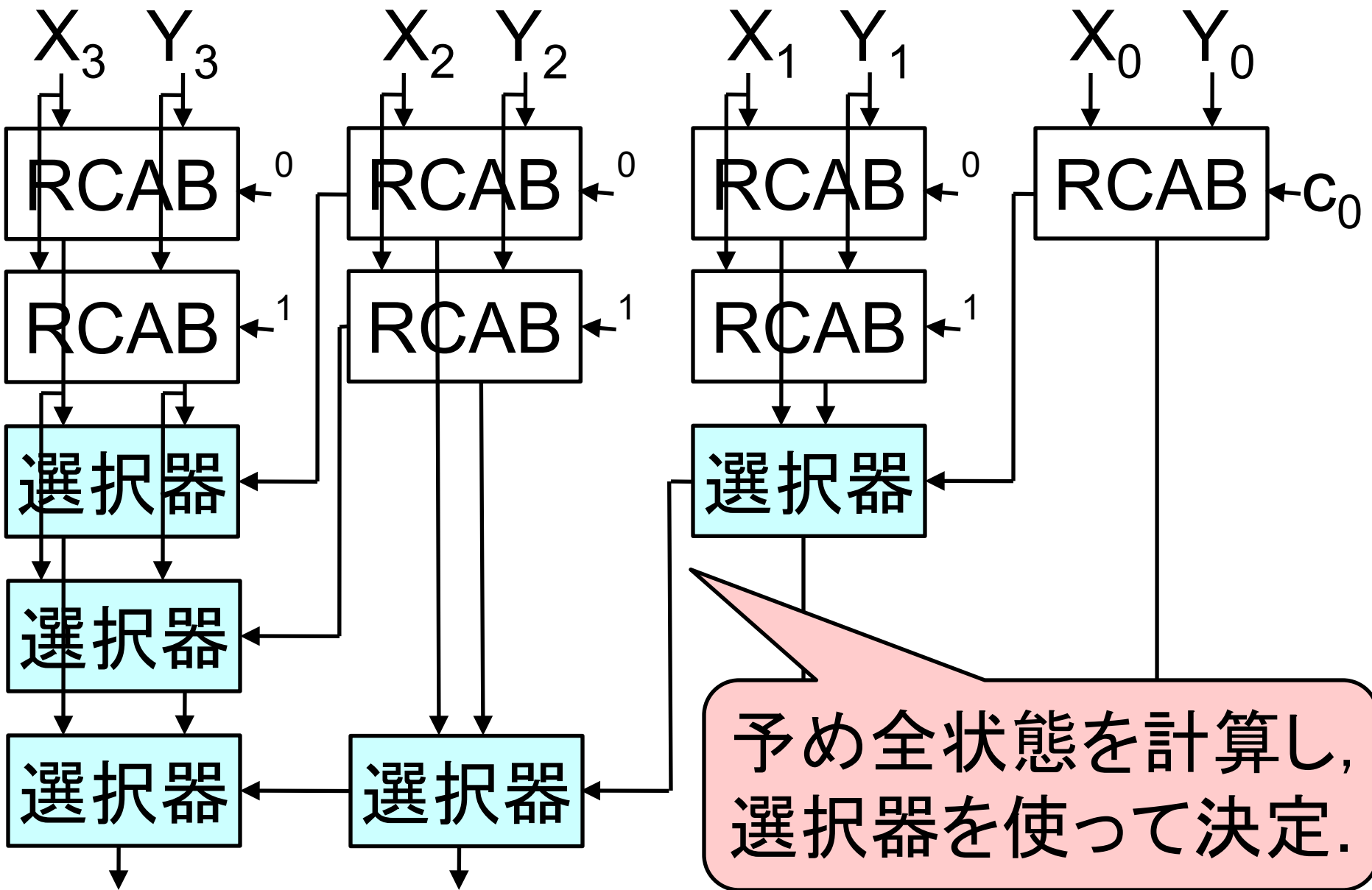
【ヒント】
桁上げがブロック内を伝播するも時間が必要

桁上げ飛越し加算器（CBA）

1. 英名： Carry Bypass (or Skip) Adder
2. RCAに桁上げ飛越し回路を付加
3. 計算時間O(√n)～O(n)，回路量 O(n)

回路構造をいちいち考えるのは大変！
高速化しつつ単純な構造にしよう。⇒ CSA

桁上げ選択加算器（CSA）



桁上げ選択加算器（CSA）

1. 英名： Carry Select Adder

2. RCAを複数用意し全状態を計算

3. 計算時間O(√n)-O(n)，回路量 O(n)

選択回路を1段ずつではなく多段にし，
効率を上げれば良いのでは ⇒ CSA

条件求和加算器（CSA）

I	7		6		5		4		3		2		1		0	
X	0		0		0		1		0		1		0		1	
Y	1		0		1		0		1		1		0		1	
	c	s	c	s	c	s	c	s	c	s	c	s	c	s	c	s
c_{in}=0	0	1	0	0	0	1	0	1	0	1	1	0	0	0	1	0
c_{in}=1	1	0	0	1	1	0	1	0	1	0	1	1	0	1		
c_{in}=0	0	1		0	0	1		1	1	0		0	0	1		0
c_{in}=1	0	1		1	1	0		0	1	0		1				
c_{in}=0	0	1		0		1		1	1	0		0		1		0
c_{in}=1	0	1		1		0		0								
S	0	1		1		0		0		0		0		1		0

条件求和加算器（CSA）

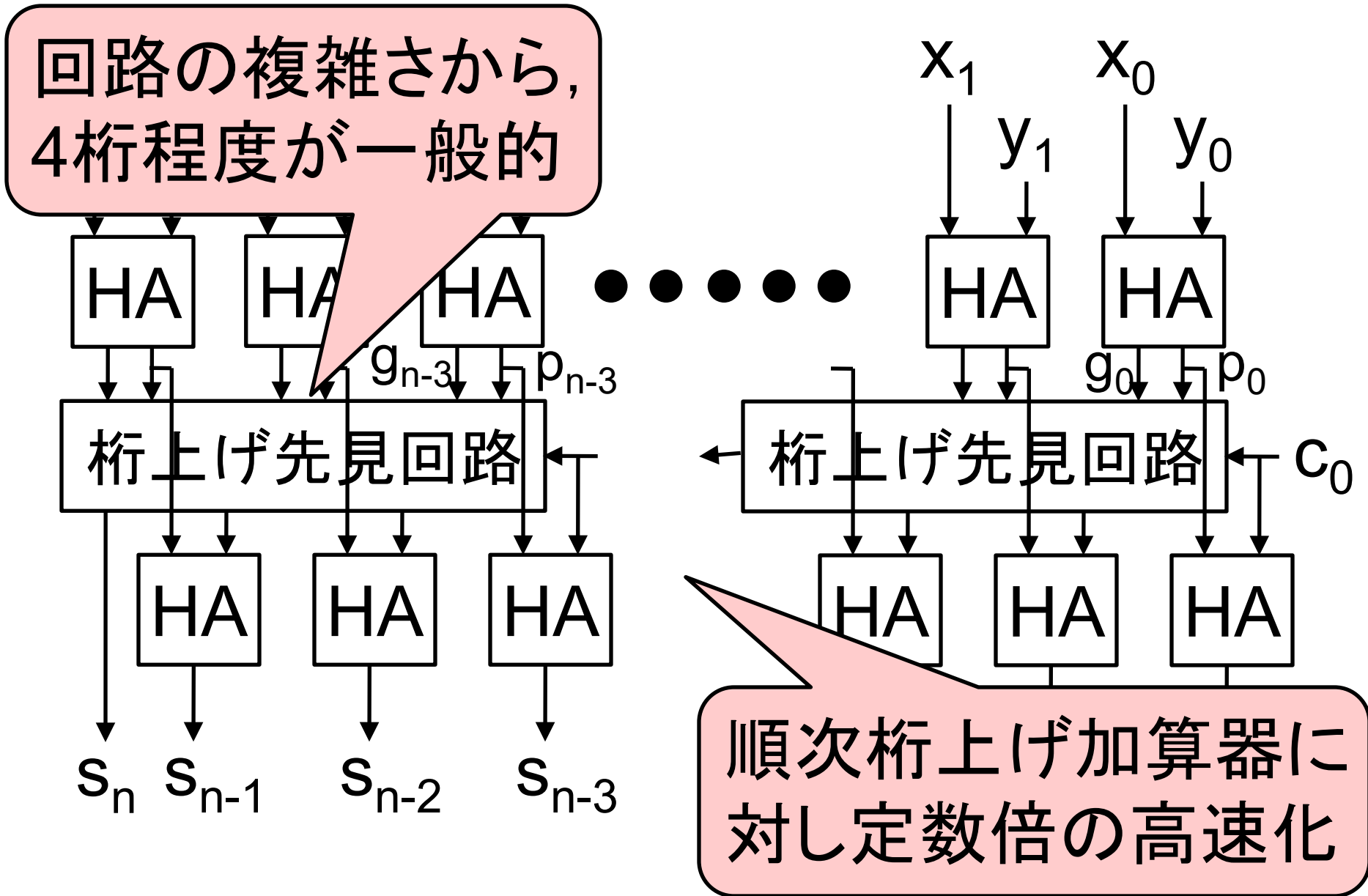
1. 英名： Conditional Sum Adder

2. 桁上げ選択加算の発展．木構造をとる．

3. 計算時間O(log n)，回路量は O(n)

回路を多段にするのは非常に有効な方法．
計算時間を飛躍的に削減できることがある．

桁上げ先見加算器（CLHA）



桁上げ先見加算器（CLHA）

1. 第 i 桁において，$g_i = x_i \cdot y_i, p_i = x_i \oplus y_i$

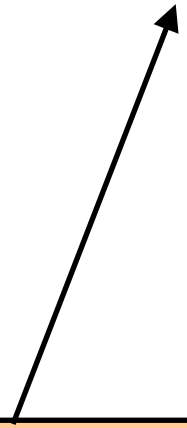
2. $c_{i+1} = g_i \vee p_i c_i = g_i \vee p_i g_{i-1} \vee p_i p_{i-1} c_{i-1}$

3. 全桁の桁上がりを計算するのは難しい．
   桁上げ先見ブロックを作成するのが一般的

$$G_h = g_{j+k-1} \vee p_{j+k-1} g_{j+k-2} \vee \cdots \vee p_{j+k-1} p_{j+k-2} \cdots p_{j+1} g_j$$

桁上げ先見加算器（CLHA）

1. 英名： Carry Lock Ahead Adder
2. 桁上げ伝搬を早くすることで全体を高速化
3. 計算時間O(log n)，回路量は O(n)



回路を多段にするのは非常に有効な方法.
計算時間を飛躍的に削減できることがある.

プレフィックス加算

1. 二桁からなるブロックの桁上げ生成条件

$$g_{i+1,i} = g_{i+1} \vee p_{i+1} \cdot g_i$$

2. 二桁からなるブロックの桁

$$p_{i+1,i} = p_{i+1} \cdot p_i$$

＊は，結合則は成立するが，交換則は不成立

3. 以下の計算を定義する

$$(g_{i+1,i}, p_{i+1,i}) = (g_{i+1} \cdot p_{i+1}) * (g_i \cdot p_i)$$

並列プレフィックス加算

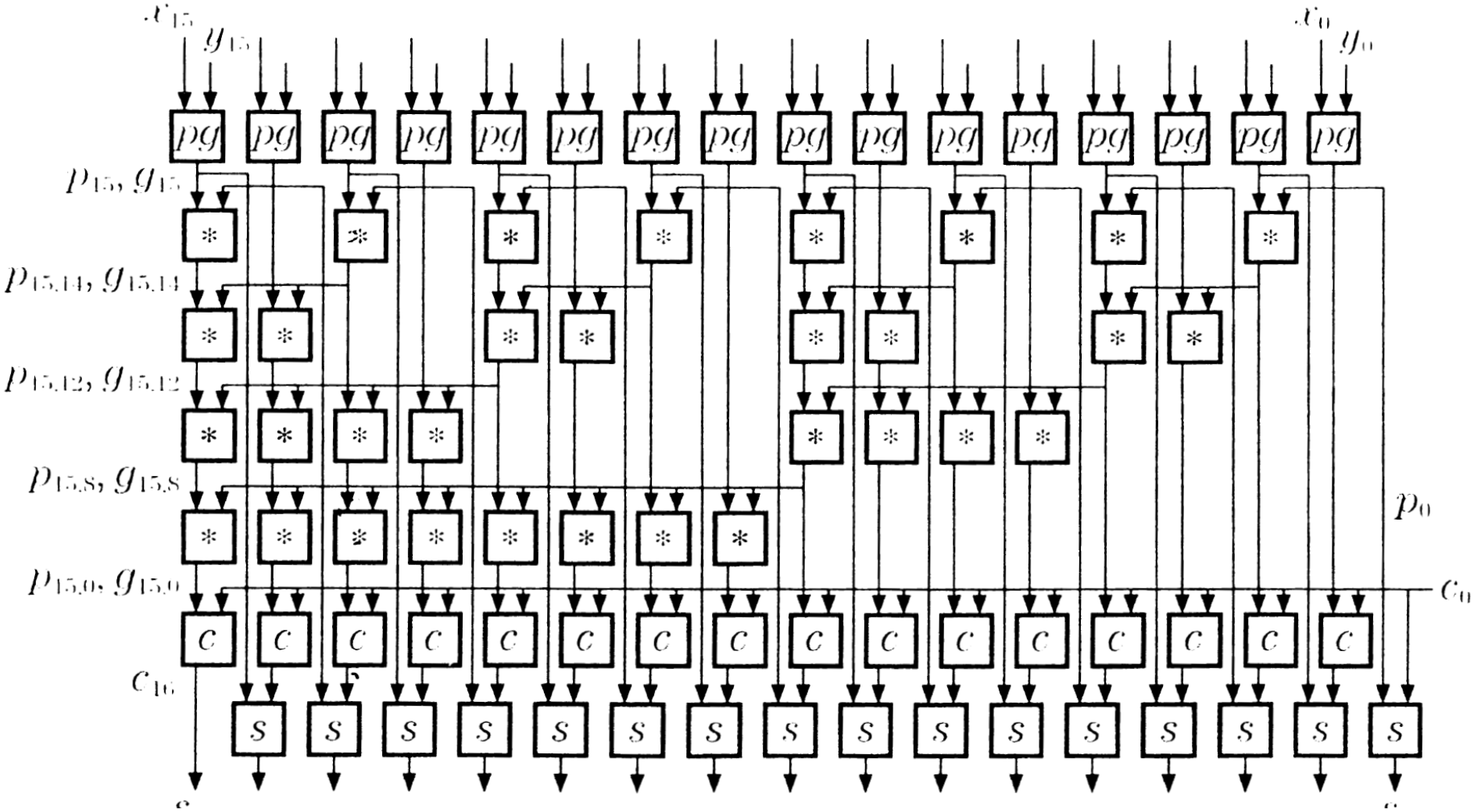
1. 第 i 桁から第 j 桁のプレフィックス計算は,

$$(g_{j,i}, p_{j,i}) = (g_j \cdot p_j) * \cdots * (g_{i+1} \cdot p_{i+1}) * (g_i \cdot p_i)$$

2. ＊は結合則が成立 ⇒ 繰返し計算を並列化

3. 以上を回路化（並列＋プレフィックス計算）

並列プレフィックス加算器



並列プレフィックス加算器

1. 英名： Parallel Prefix Adder
2. プレフィックス計算の並列化により高速化
3. 計算時間O(log n), 回路量O(n)- O(nlog n)

加算器のアルゴリズムを使ってO(log n)まで計算時間を短縮できる.

加算器のまとめ

アルゴリズム	計算時間	回路量
順次桁上げ	O(n)	O(n)
桁上げ飛越し	O(√n)～O(n)	O(n)
桁上げ選択	O(√n)～O(n)	O(n)
桁上げ先見	O(log n)	O(n)
並列プレフィックス	O(log n)	O(n)～O(n log n)