論理設計
東野担当６回目
授業スライド
１１月１１日

基礎工学部情報科学科　東野輝夫

授業時間変更＆中間試験のお知らせ

- 長谷川先生のご都合で，長谷川先生担当の水曜３限の計算機言語の授業と水曜４限の東野の論理設計の授業を下記のように交換して実施します．
- １１月２５日（水）
  - ３限　計算機言語　→　論理設計　の授業に変更
  - ４限　論理設計　　（この日は２コマ続けて論理設計の授業を実施）
- １２月　９日（水）
  - ３限　計算機言語　（この日は２コマ続けて計算機言語の授業を実施）
  - ４限　論理設計　→　計算機言語　の授業に変更

- 東野担当の論理設計の中間試験について
  - １１月２５日（水）の４限（15:10 開始）に東野担当の論理設計の中間試験を実施します．
  - この日は３限に普通の授業を実施し，４限に試験を実施します．試験はCLE上で実施します．詳細は別途連絡しますが，下記で実施予定．
    - 15:10　試験の方法を説明（CLEビデオ or Zoom）
    - 15:20　CLEに試験問題を掲示し，制限時間を決めて時間内にレポート課題提出と同じ方法で答案をuploadしてもらうことで答案回収します．

授業計画の変更

長谷川先生との授業の入れ替えに伴い，章の説明の順番を変更します

- 授業計画：東野担当の授業計画を下記のように変更します．
  1. ドントケアを含む論理関数の簡単化（６章）
  2. フリップフロップとレジスタ（１０章）
  3. 同期式順序回路（Mealy型, Moore型順序回路）（１１章）
  4. カルノー図を用いた論理関数の簡単化（１章から５章の復習）
  5. 組合せ論理回路設計、よく用いられる組み合わせ回路（７章，８章）
  6. 加減算器とALU、順序回路の簡単化（９章，１２章前半）
  7. 演習
  8. 順序回路の簡単化、カウンタ（１２章後半，１３章）
  9. 中間試験（１章～１１章）
  10. ＩＣを用いた順序回路の実現（１５章）
  11. 演習
  12. ＣＰＵの設計（付録）
  13. ＣＰＵの設計，演習
  14. 乗算器と除算器（１４章）
  15. 期末試験（１２章～１５章，付録）

８コマ目の授業を11/25の３限に実施
９コマ目の試験を11/25の４限に実施
中間試験の範囲を１１章までに変更
期末試験の範囲を１２章以降に変更

質問について

- メールで随時問い合わせや質問にお答えしますので，何かあれば，higashino@ist.osaka-u.ac.jp までメールで質問して下さい.
- また、時間を決めてZoomなどを用いて質問にお答えすることも可能ですので、まずはメールで疑問点や問い合わせ事項などを連絡して下さい。

お願い

本テキストや授業のビデオなどの電子ファイルを他人に転送したり、ネットへアップロードすることなどを禁止します。

著作権保護

- この授業のテキスト（教科書）や授業スライド、授業ビデオの著作権保護に努めて下さい.
- この授業のビデオやスナップショットを録画したり，それらを他の人に転送したり，インターネット上で公開したりすることを禁止します.
- この授業で利用するスライドにはオーム社の教科書の図などが含まれているので，著作権保護の観点から，この授業スライドの公開につながる行為は謹んでください.
- 来年度は CLE を使ったメディア授業でなく，対面の授業ができることを期待していますが，今年度の演習課題の解答が事前に公開されたりすると，来年度の授業で同じ演習課題が使えなくなり，授業テキストの大幅な修正が必要になるため，協力をお願いします.

OSAKA UNIVERSITY

５回目の授業の
レポート課題の解答

５回目の授業のレポート課題

レポート課題

（課題５）

- 3変数論理関数 $f_1(x_2,x_1,x_0)$ は (x_2,x_1,x_0) を３ビットの２進数（x_2が上位ビット，x_0が下位ビット）と見なした際に 1,4,5,7のとき $f_1=1$、そうでないとき $f_1=0$ とする．また，$f_2(x_2,x_1,x_0)$は、1,2,3 のとき $f_2=1$、そうでないとき $f_2=0$ とする．
  1. f_1，f_2および $f_1 \cdot f_2$ の最簡積和形をそれぞれ求めよ。
  2. 上の1.で求めた主項を用いて、 $f_1(x_2,x_1,x_0)$， $f_2(x_2,x_1,x_0)$を同時に実現する最簡多出力論理関数を求めよ。
  - カルノー図に主項を明記して， f_1，f_2および $f_1 \cdot f_2$ の最簡積和形をどのように求めたかや，なぜ解答の関数が最簡多出力論理関数になるかの理由を明記してください．
- 提出先：wordやpower pointなどで電子的に作成するか，紙に書いた解答をスマホで写真を取ったりスキャナーなどで読み取り、pdf や jpeg, gif などの形式で電子化して CLE にアップして下さい．
- 締切：11月10日(火) 23:59（次の授業の前日迄）．
- ６回目の授業は 11月11日(水) の15:10 から実施します．授業のビデオは当日の12:30以降に視聴できるようになります．

5回目の授業のレポート課題解答

- f_1，f_2および $f_1 \cdot f_2$ の最簡積和形はそれぞれ下記の通り（すべての主項が必須項になっている）． f_1，f_2の論理積５つ，論理和２個.
  - $f_1 = x_2\bar{x}_1 \vee x_2x_0 \vee \bar{x}_1x_0$
  - $f_2 = \bar{x}_2x_0 \vee \bar{x}_2x_1$
  - $f_1 \cdot f_2 = \bar{x}_2\bar{x}_1x_0$
- 最簡多出力論理関数は下記の通り．$\bar{x}_2\bar{x}_1x_0$ が f_1，f_2で共用されている.
  - $f_1 = x_2\bar{x}_1 \vee x_2x_0 \vee \bar{x}_2\bar{x}_1x_0$
  - $f_2 = \bar{x}_2x_1 \vee \bar{x}_2\bar{x}_1x_0$
  - f_1，f_2の論理積４つ，論理和２個.

f_1	x_1x_0					f_2	x_1x_0			
x_2	00	01	11	10		x_2	00	01	11	10
0	0	1	0	0		0	0	1	1	1
1	1	1	1	0		1	0	0	0	0

f_1f_2	x_1x_0			
x_2	00	01	11	10
0	0	1	0	0
1	0	0	0	0

第９章
加減算器とALU

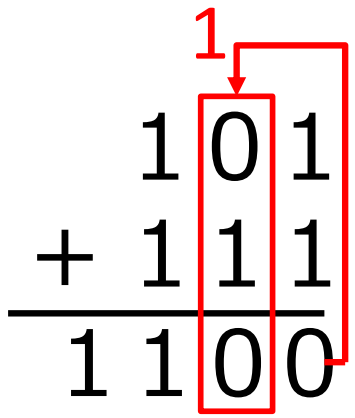
第９章　加減算器とALU

- この章のねらい

9章 加減算器とALU

本章では，コンピュータでの算術演算と論理演算を実行する基本的な演算器について説明する．まず，算術演算の基本演算器である加算器の構成方法について説明する．加算器の代表的な構成方法として，逐次桁上げ加算方式と桁上げ先見加算方式の２種類の演算器について説明する．次に，加減算器および２進化 10 進加算器について説明する．最後に算術論理演算ユニット（ALU）の構成方法について説明する．

全加算器



- 下位ビットからの桁上げを考慮した場合の 1 ビットの加算器を 全加算器（full adder: FA）と呼ぶ. 下位ビットからの桁上げを変数 z で表し，半加算器と同様に 1 ビットの 2 進数 x と y および下位桁からの桁上げ z との和を 2c + s で表すと，c や s の値は 表1·4 に示す真理値表のようになる.
- c の値が 1 になるのは，x, y, z の値のうちの二つ以上が 1 のときに限るので

  $c = (x \cdot y) \vee (y \cdot z) \vee (z \cdot x)$

  と論理式で表すことができる.
- 同様に s の値が 1 になるのは，x と y と z の値のうちの奇数個が 1 であるときに限るので

  $s = (x \cdot \overline{y} \cdot \overline{z}) \vee (\overline{x} \cdot y \cdot \overline{z}) \vee (\overline{x} \cdot \overline{y} \cdot z) \vee (x \cdot y \cdot z)$

  $= x \oplus y \oplus z$

  と論理式で表すことができる.

表 1・4　全加算器の真理値表

x	y	z	c	s
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

全加算器

- 半加算器と同様，全加算器も論理演算記号を用いて 図1·5 のように表現でき，全加算器そのものを 図1·6 のような論理演算記号を用いて表現することもある．さらに，全加算器は半加算器を用いて 図1·7 のように表現することも可能である．

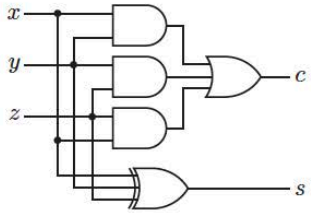


図 1・5　論理ゲートを用いた全加算器の実現

表 1・4　全加算器の真理値表

x	y	z	c	s
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

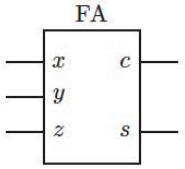


図 1・6　全加算器の論理記号

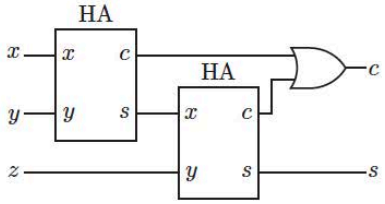


図 1・7　半加算器と論理ゲートを用いた全加算器の実現

全加算器

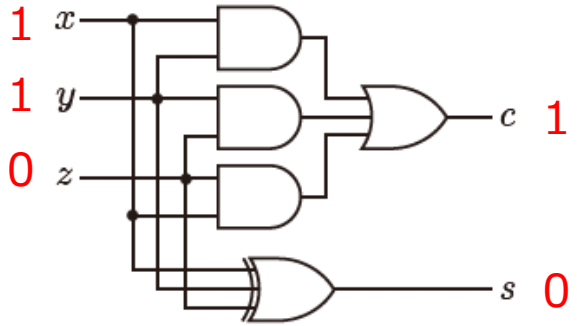


図 1・5　論理ゲートを用いた全加算器の実現

図 1・5　論理ゲートを用いた全加算器の実現

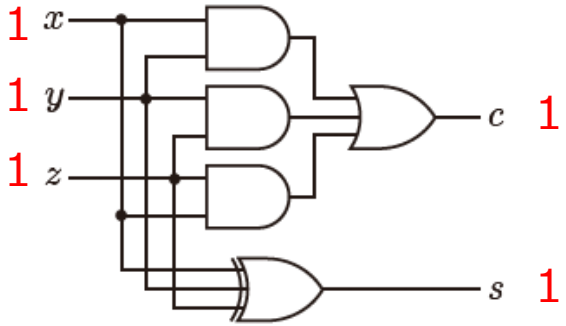


表 1・4　全加算器の真理値表

x	y	z	c	s
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

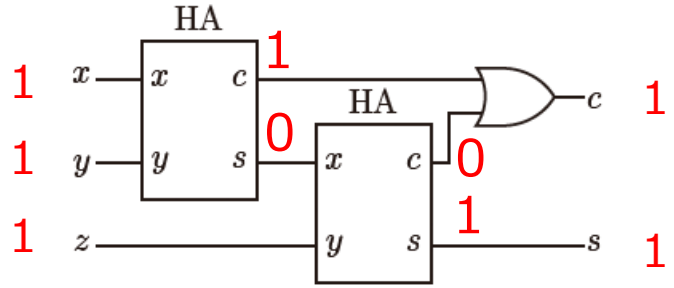


図 1・7　半加算器と論理ゲートを用いた全加算器の実現

４ビットの全加算器（逐次桁上げ加算器）

逐次桁上げ加算器

- 全加算器は最下位桁（least significant bit: LSB）からの桁上げを考慮して最上位桁（most significant bit: MSB）まで１ビットずつ加算を繰り返すことで実現できるので，全加算器を 図1･8 のように接続すると，複数ビットの2 進数の加算演算を実現することができる．このような加算器を逐次桁上げ加算器（ripple carry adder: RCA）と呼ぶ．

$(x_3,x_2,x_1,x_0)=(1,0,1,1)$ (11)
$(y_3,y_2,y_1,y_0)=(1,0,1,0)$ (10)

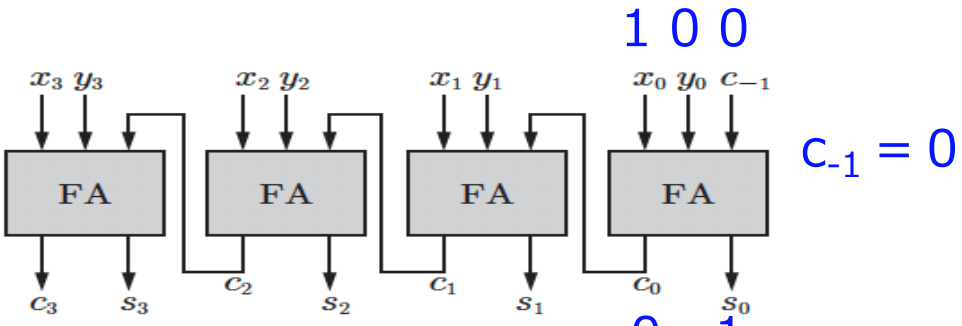


図 1・8　4 ビット加算器

```
    1 0 1 1   (11)
+   1 0 1 0   (10)
  1 0 1 0 1   (21)
```

c_3 $s_3s_2s_1s_0$

４ビットの全加算器（逐次桁上げ加算器）

逐次桁上げ加算器

- 全加算器は最下位桁（least significant bit: LSB）からの桁上げを考慮して最上位桁（most significant bit: MSB）まで１ビットずつ加算を繰り返すことで実現できるので，全加算器を 図1･8 のように接続すると，複数ビットの2 進数の加算演算を実現することができる．このような加算器を逐次桁上げ加算器（ripple carry adder: RCA）と呼ぶ．

$(x_3,x_2,x_1,x_0)=(1,0,1,1)$ (11)
$(y_3,y_2,y_1,y_0)=(1,0,1,0)$ (10)

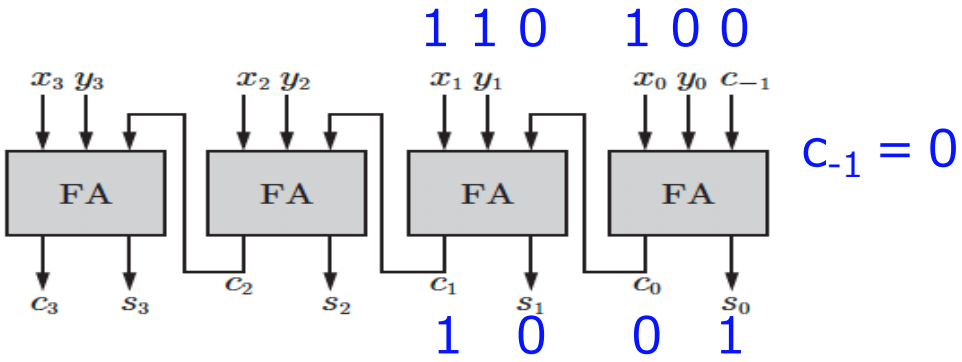


図 1・8　4 ビット加算器

```
    1 0 1 1   (11)
+   1 0 1 0   (10)
  1 0 1 0 1   (21)
```

c_3 $s_3s_2s_1s_0$

４ビットの全加算器（逐次桁上げ加算器）

逐次桁上げ加算器

- 全加算器は最下位桁（least significant bit: LSB）からの桁上げを考慮して最上位桁（most significant bit: MSB）まで１ビットずつ加算を繰り返すことで実現できるので，全加算器を 図1･8 のように接続すると，複数ビットの2 進数の加算演算を実現することができる．このような加算器を逐次桁上げ加算器（ripple carry adder: RCA）と呼ぶ．

$(x_3,x_2,x_1,x_0)=(1,0,1,1)$ (11)
$(y_3,y_2,y_1,y_0)=(1,0,1,0)$ (10)

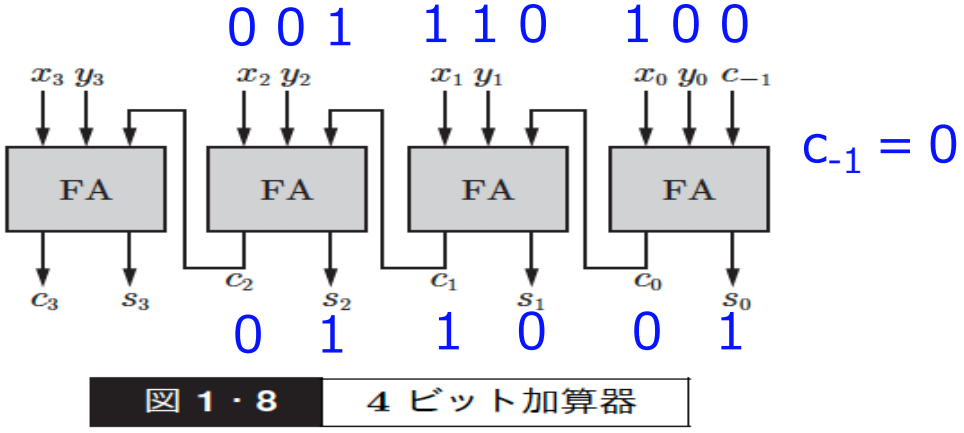


図 1・8　4 ビット加算器

```
  1 0 1 1   (11)
+ 1 0 1 0   (10)
-----------------
1 0 1 0 1   (21)
```

c_3　$s_3s_2s_1s_0$

４ビットの全加算器（逐次桁上げ加算器）

逐次桁上げ加算器

- 全加算器は最下位桁（least significant bit: LSB）からの桁上げを考慮して最上位桁（most significant bit: MSB）まで１ビットずつ加算を繰り返すことで実現できるので，全加算器を 図1･8 のように接続すると，複数ビットの2 進数の加算演算を実現することができる．このような加算器を逐次桁上げ加算器（ripple carry adder: RCA）と呼ぶ．

$(x_3,x_2,x_1,x_0)=(1,0,1,1)$ (11)
$(y_3,y_2,y_1,y_0)=(1,0,1,0)$ (10)

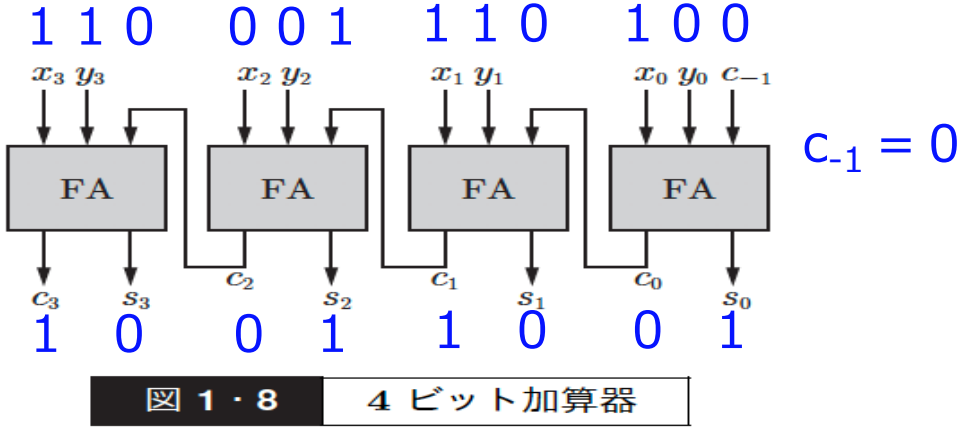


図 1・8　4 ビット加算器

```
    1 0 1 1   (11)
+   1 0 1 0   (10)
  1 0 1 0 1   (21)
```

c_3 $s_3s_2s_1s_0$

逐次桁上げ加算器

- 逐次桁上げ加算器（$X+Y=<c_{n-1},S>$）
  - $X = (x_{n-1}, x_{n-2},\cdots, x_0)$
  - $Y = (y_{n-1}, y_{n-2},\cdots, y_0)$
  - $S = (s_{n-1}, s_{n-2},\cdots, s_0)$　および　最上位桁からの桁上げ出力 c_{n-1}

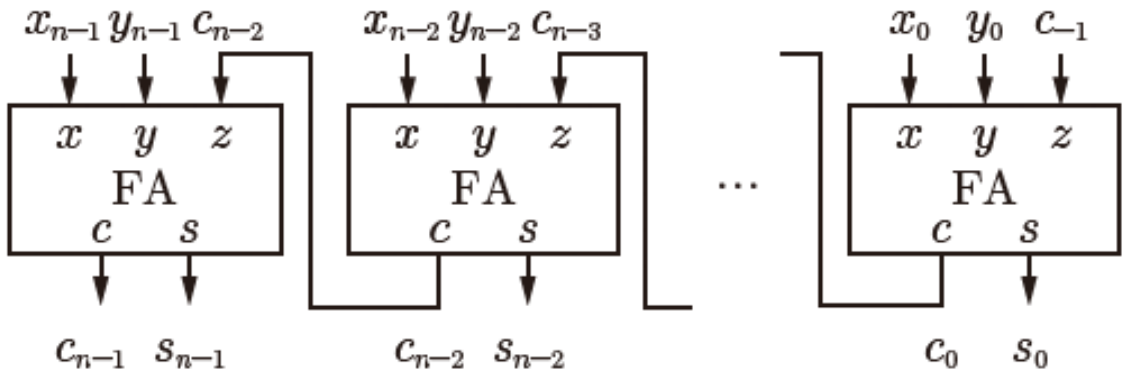
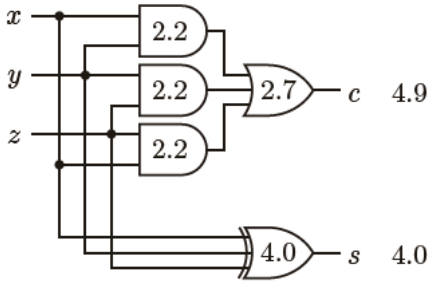


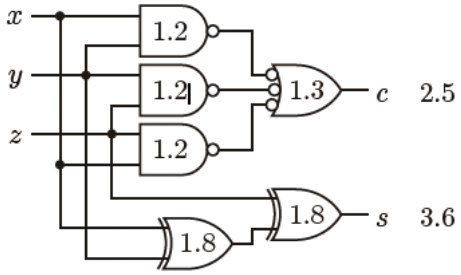
図 9・2　逐次桁上げ加算器

逐次桁上げ加算器の遅延時間

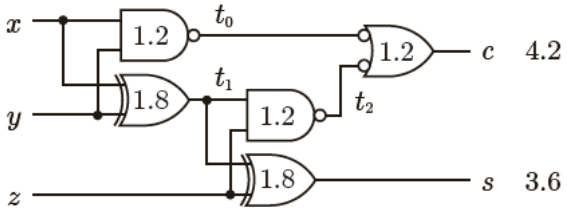
- 逐次桁上げ加算器（$X+Y=<c_{n-1},S>$）
  - $X = (x_{n-1}, x_{n-2},\cdots, x_0)$
  - $Y = (y_{n-1}, y_{n-2},\cdots, y_0)$
  - $S = (s_{n-1}, s_{n-2},\cdots, s_0)$　および　最上位桁からの桁上げ出力 c_{n-1}



（a） AND, OR, XOR ゲートを用いた実装



（b） NAND, XOR ゲートを用いた実装(1)



（c） NAND, XOR ゲートを用いた実装(2)

図 9・1　全加算器

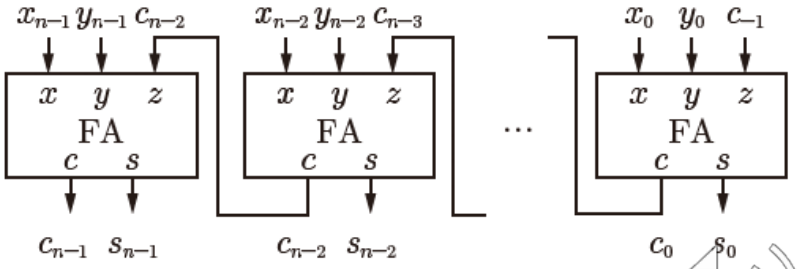
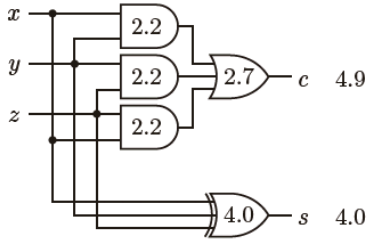


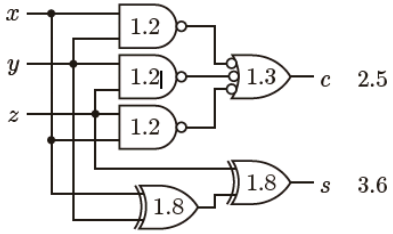
図 9・2　逐次桁上げ加算器

逐次桁上げ加算器の遅延時間

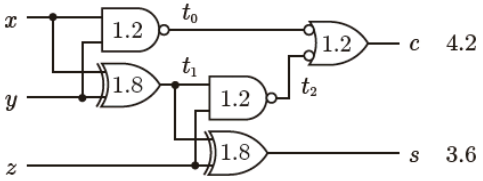
- 全加算器の遅延時間：入力されてから出力されるまでの遅延時間の最大値となる経路（最大遅延経路：critical path）の遅延時間を回路の遅延時間と考える
- 逐次桁上げ加算器（X+Y=<c_{n-1},S>）の遅延時間：１ビットの全加算器の遅延時間の n 倍（*O*(n)の遅延時間）



（a） AND, OR, XOR ゲートを用いた実装



（b） NAND, XOR ゲートを用いた実装(1)



（c） NAND, XOR ゲートを用いた実装(2)

図 9・1 全加算器

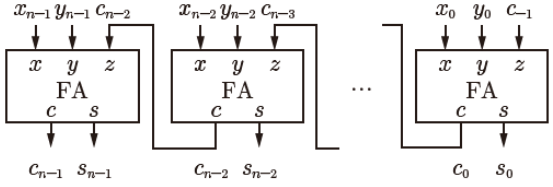


図 9・2 逐次桁上げ加算器

- 例題９・１
  - 図９・１(b)の遅延時間？
    （NOTの遅延時間をTとする）
  - c : 1.2T + 1.3T = 2.5T
  - s : 1.8T + 1.8T = 3.6T
  - ちなみに図９・１(a)の場合
  - c : 2.2T + 2.7T = 4.9T
  - s : 4.0T
  - 図９・１(c)の場合
  - c : 1.8T + 1.2T + 1.2T = 4.2T
  - s : 1.8T + 1.8T = 3.6T

桁上げ先見加算器

- 逐次桁上げ加算器の遅延時間：$O(n)$の遅延時間
- 桁上げ先見加算器の遅延時間：$O(\log_2 n)$の遅延時間
  - 桁上げ生成関数と桁上げ伝搬関数を先に計算する．

  - 第 i 桁目の桁上げ信号 c_i $(i > 0)$ は，次の式で計算される．

    $c_i = (x_i \cdot y_i) \vee (x_i \cdot c_{i-1}) \vee (y_i \cdot c_{i-1})$

    $= (x_i \cdot y_i) \vee ((x_i \oplus y_i) \cdot c_{i-1})$
  - この式の中で，$g(x_i,y_i) = x_i \cdot y_i$ は桁上げ生成関数（carry generation function），$p(x_i,y_i) = x_i \oplus y_i$ は桁上げ伝搬関数（carry propagation function）と呼ばれる．
  - 桁上げ生成関数はその桁のx_i,y_iのみで桁上げが生じるかどうかを表す関数
  - 桁上げ伝搬関数は下位桁から桁上げがあった場合に，上位に桁上げが生じるかどうかを表す関数

４ビット桁上げ先見加算器

４ビット桁上げ先見加算器（CLA-4）

- ４ビットでの桁上げ生成関数Gと桁上げ伝搬関数Pを計算する関数を設ける
  - Gは(x_3,x_2,x_1,x_0)と(y_3,y_2,y_1,y_0)のみで桁上げが起こるかどうかを表す関数
  - Pは下からの桁上げがあった場合に桁上げが起こるかどうかを表す関数

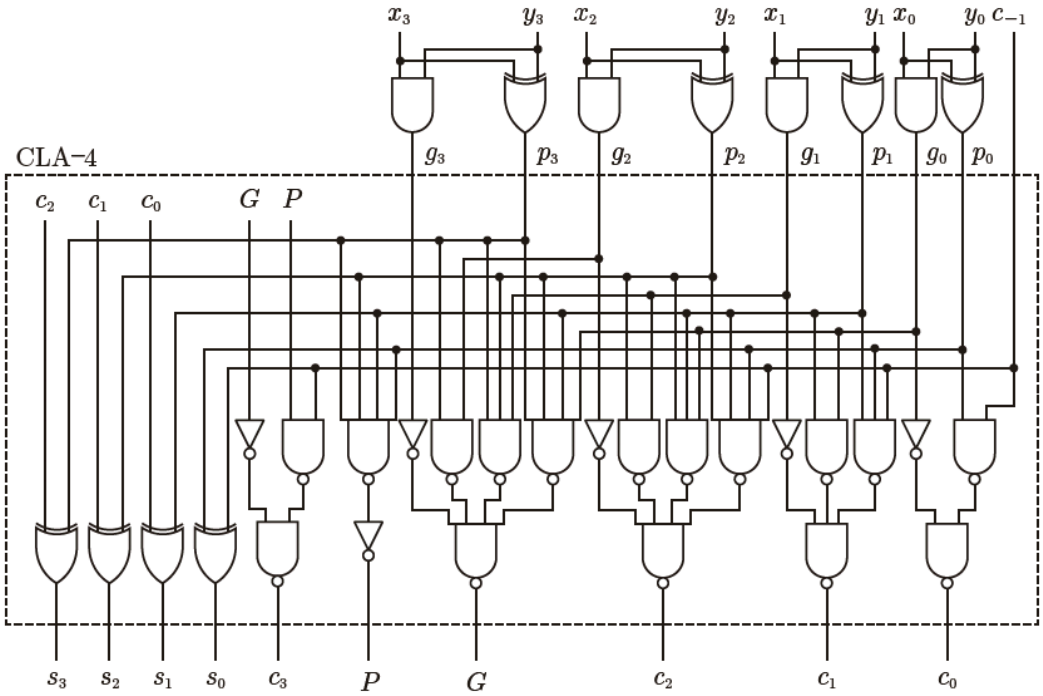


図 9・3　４ビット桁上げ先見加算器の論理回路図

$g(x_i,y_i) = x_i \cdot y_i$ は
桁上げ生成関数
$p(x_i,y_i) = x_i \oplus y_i$ は
桁上げ伝搬関数

$$c_i = g(x_i,y_i) \vee p(x_i,y_i) \cdot c_{i-1}$$

４ビット桁上げ先見加算器

４ビット桁上げ先見加算器（CLA-4）

- $c_0 = g_0 \vee (p_0 \cdot c_{-1})$
- $c_1 = g_1 \vee (p_1 \cdot c_0)$
  $= g_1 \vee \{p_1 \cdot (g_0 \vee p_0 \cdot c_{-1})\}$
  $= g_1 \vee (p_1 \cdot g_0) \vee (p_1 \cdot p_0 \cdot c_{-1})$
- $c_2 = g_2 \vee (p_2 \cdot c_1)$
  $= g_2 \vee \{p_2 \cdot (g_1 \vee p_1 \cdot g_0 \vee p_1 \cdot p_0 \cdot c_{-1})\}$
  $= g_2 \vee (p_2 \cdot g_1) \vee (p_2 \cdot p_1 \cdot g_0) \vee (p_2 \cdot p_1 \cdot p_0 \cdot c_{-1})$
- $c_3 = g_3 \vee (p_3 \cdot c_2)$
  $= g_3 \vee \{p_3 \cdot (g_2 \vee p_2 \cdot g_1 \vee p_2 \cdot p_1 \cdot g_0 \vee p_2 \cdot p_1 \cdot p_0 \cdot c_{-1})\}$
  $= g_3 \vee (p_3 \cdot g_2) \vee (p_3 \cdot p_2 \cdot g_1) \vee (p_3 \cdot p_2 \cdot p_1 \cdot g_0)$
  $\vee (p_3 \cdot p_2 \cdot p_1 \cdot p_0 \cdot c_{-1})$
  $= \{g_3 \vee (p_3 \cdot g_2) \vee (p_3 \cdot p_2 \cdot g_1) \vee (p_3 \cdot p_2 \cdot p_1 \cdot g_0)\}$
  $\vee \{p_3 \cdot p_2 \cdot p_1 \cdot p_0\} \cdot c_{-1}) = G \vee P \cdot c_{-1}$
- $G = \{g_3 \vee (p_3 \cdot g_2) \vee (p_3 \cdot p_2 \cdot g_1) \vee (p_3 \cdot p_2 \cdot p_1 \cdot g_0)\}$　積和形
- $P = \{p_3 \cdot p_2 \cdot p_1 \cdot p_0\}$　論理積

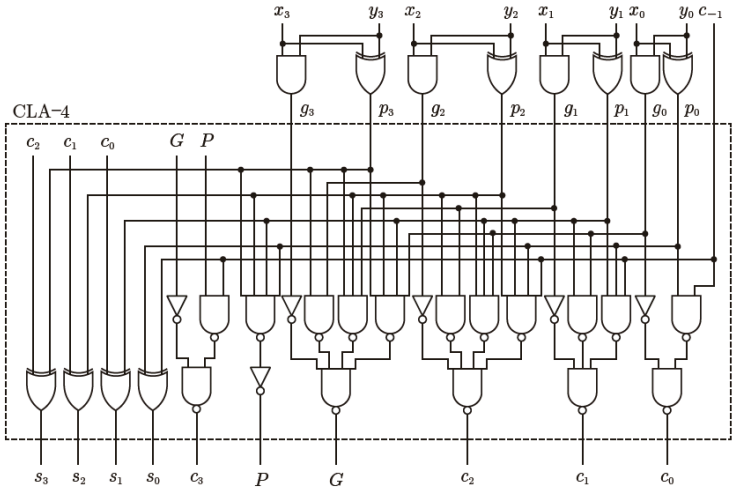


図９・３　４ビット桁上げ先見加算器の論理回路図

４ビット桁上げ先見加算器

$g(x_i,y_i) = x_i \cdot y_i$ は桁上げ生成関数

$p(x_i,y_i) = x_i \oplus y_i$ は桁上げ伝搬関数

４ビット桁上げ先見加算器（CLA-4）

- $c_3 = G \vee P \cdot c_{-1}$
- $G = \{g_3 \vee (p_3 \cdot g_2) \vee (p_3 \cdot p_2 \cdot g_1) \vee (p_3 \cdot p_2 \cdot p_1 \cdot g_0)\}$ 積和形
- $P = \{p_3 \cdot p_2 \cdot p_1 \cdot p_0\}$ 論理積
- ・, ∨, ¬, ⊕ の遅延時間を⊿と簡単化すると

- $c_0 = g_0 \vee (p_0 \cdot c_{-1})$
- $c_1 = g_1 \vee (p_1 \cdot g_0) \vee (p_1 \cdot p_0 \cdot c_{-1})$
- $c_2 = g_2 \vee (p_2 \cdot g_1) \vee (p_2 \cdot p_1 \cdot g_0) \vee (p_2 \cdot p_1 \cdot p_0 \cdot c_{-1})$

遅延時間：2⊿

- $s_3 = p_3 \oplus c_2$
- $s_2 = p_2 \oplus c_1$
- $s_1 = p_1 \oplus c_0$
- $s_0 = p_0 \oplus c_{-1}$

遅延時間：さらに⊿

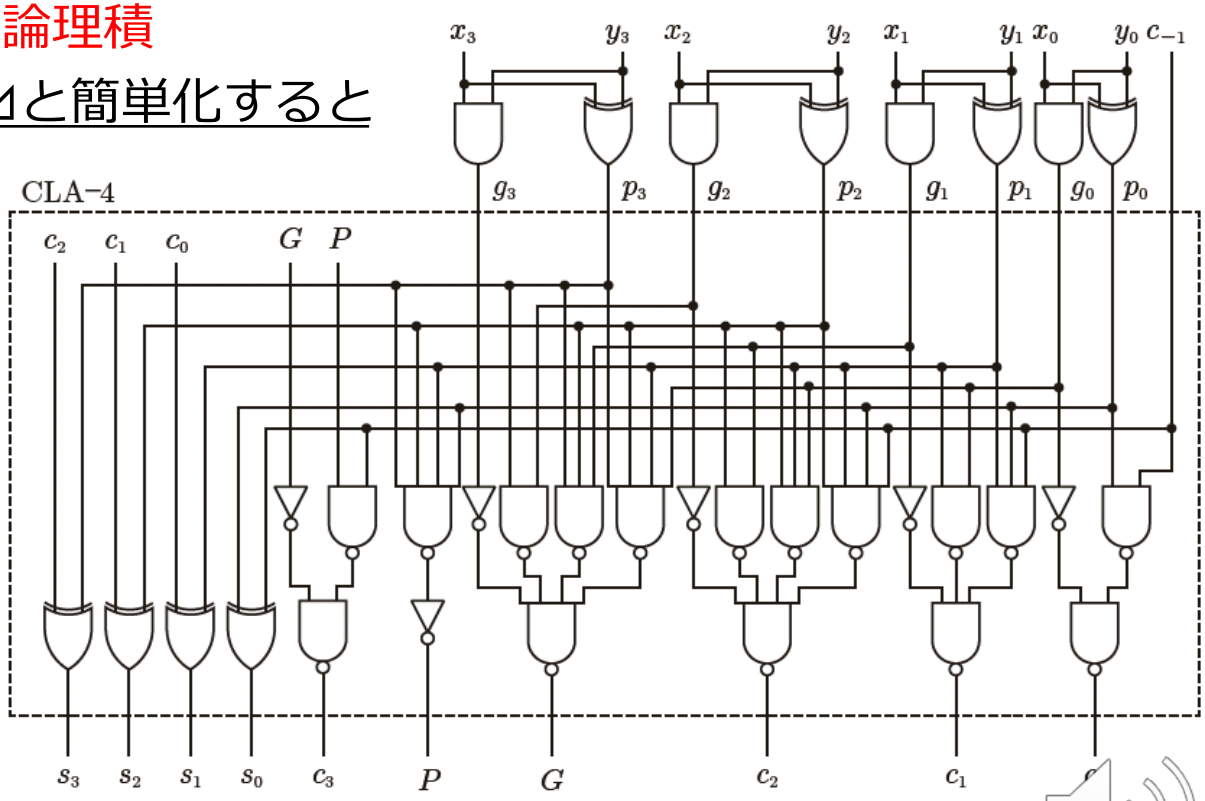


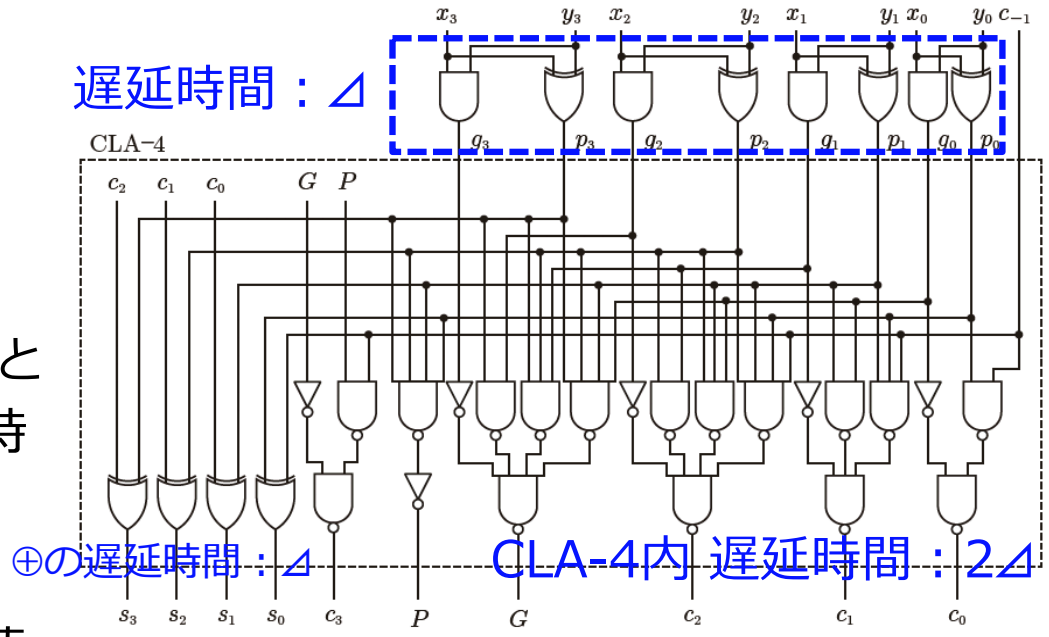
図 9・3 ４ビット桁上げ先見加算器の論理回路図

４ビット桁上げ先見加算器の遅延時間

$g(x_i,y_i) = x_i \cdot y_i$ は桁上げ生成関数
$p(x_i,y_i) = x_i \oplus y_i$ は桁上げ伝搬関数

４ビット桁上げ先見加算器（CLA-4）

- 入力 $(x_{m-1},\cdots,x_0)$, $(y_{m-1},\cdots,y_0)$, c_{-1} が入力されてから和出力 $(s_{m-1},\cdots,s_0)$ が出力されるまでの遅延時間を α_m，入力 $(x_{m-1},\cdots,x_0)$, $(y_{m-1},\cdots,y_0)$,が入力されてから G,P が出力されるまでの遅延時間を β_m，桁上げ入力 c_{-1} が入力されて和出力 $(s_{m-1},\cdots,s_0)$ の値が確定するまでの遅延時間を γ_m とする.
- ・, ∨, ¬, ⊕ の遅延時間：⊿
  - α_4, β_4, γ_4 ?
  - Gの遅延時間：⊿+2⊿=3⊿
  - Pの遅延時間：⊿+2⊿=3⊿
  - $\beta_4 = 3\Delta$
  - s_i の値：$c_{i\ (i=0,1,2)}$の値が決まると $s_i = p_i \oplus c_{i-1}$ (i=0,1,2,3)で⊿時間後にそれらの値が決まるので $\alpha_4 = 4\Delta$
  - c_{-1}の値が入力されてから s_i の値が決まるまでの時間（$c_{i\ (i=0,1,2)}$の値が決まるまで2⊿時間かかり、これに $s_i = p_i \oplus c_{i-1}$ の計算に必要な時間⊿を加える）：$\gamma_4 = 3\Delta$



１６ビット桁上げ先見加算器の遅延時間

$g(x_i,y_i) = x_i \cdot y_i$ は桁上げ生成関数
$p(x_i,y_i) = x_i \oplus y_i$ は桁上げ伝搬関数

16ビット桁上げ先見加算器（CLA-16）

- {CLA-4}４個＋CLA-4の２段回路で実現
- 入力 $(x_{m-1},\cdots,x_0)$, $(y_{m-1},\cdots,y_0)$, c_{-1} が入力されてから和出力 $(s_{m-1},\cdots,s_0)$ が出力されるまでの遅延時間を α_m，入力 $(x_{m-1},\cdots,x_0)$, $(y_{m-1},\cdots,y_0)$,が入力されてから G,P が出力されるまでの遅延時間を β_m とする．また，桁上げ入力 c_{-1} が入力されて和出力 $(s_{m-1},\cdots,s_0)$ の値が確定するまでの遅延時間を γ_m とする．
- ・, ∨, ¬, ⊕ の遅延時間：⊿
  - α_4 ：4⊿
  - $\beta_4 = \gamma_4 = 3$⊿
- 下段のCLA-4の c_3, c_7, c_{11}の値が決まらないと、上段のc_3, c_7, c_{11} の値が決まらない！
  - $\alpha_{16} = \beta_4+2$⊿$+\gamma_4 = 8$⊿
  - $\beta_{16} = \beta_4+2$⊿$= 5$⊿
  - $\gamma_{16} = \gamma_4+2$⊿$= 5$⊿

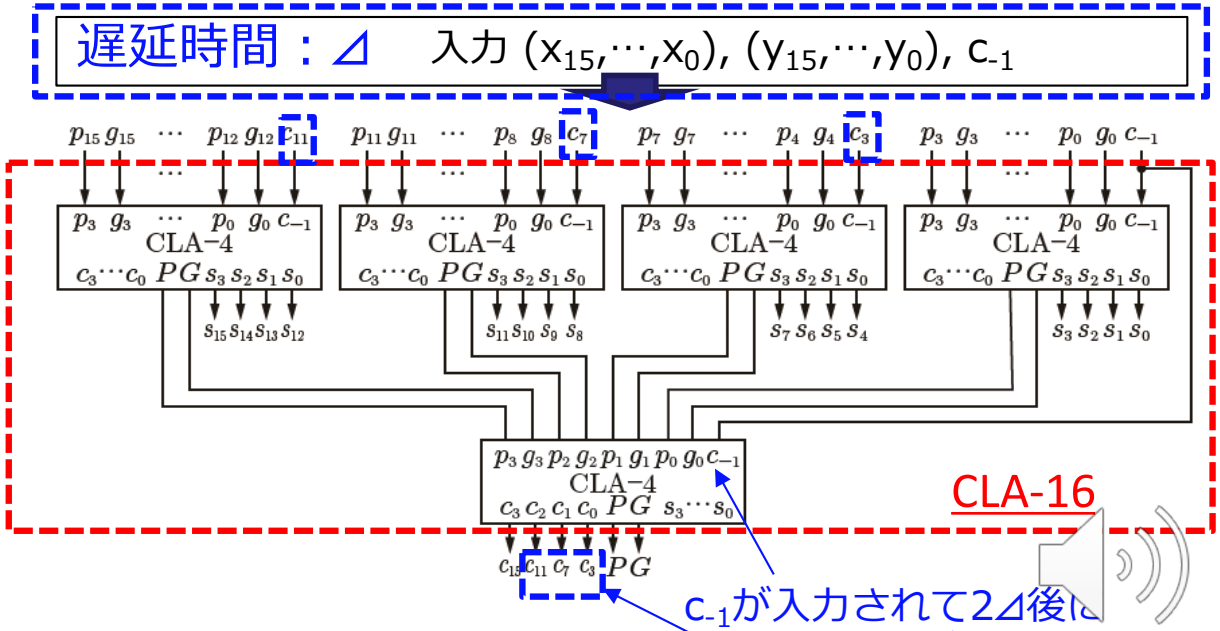


図９・４　16 ビット桁上げ先見加算器の構成方法

６４ビット桁上げ先見加算器の遅延時間

OSAKA UNIVERSITY

64ビット桁上げ先見加算器（CLA-64）

- {CLA-16}４個＋CLA-4の２段回路で実現（・, ∨, ¬, ⊕ の遅延時：⊿）
  - $\alpha_{4m} = \beta_m + 2\Delta + \gamma_m$
  - $\beta_{4m} = \beta_m + 2\Delta$
  - $\gamma_{4m} = \gamma_m + 2\Delta$
  - $\alpha_{64} = 5\Delta+2\Delta+5\Delta = 12\Delta$
  - $\beta_{64} = \gamma_{64} = 5\Delta+2\Delta = 7\Delta$
  - $\alpha_4 = 4\Delta$
  - $\beta_4 = \gamma_4 = 3\Delta$
  - $\alpha_{16} = 8\Delta$
  - $\beta_{16} = \gamma_{16} = 5\Delta$
  - $\alpha_{64} = 12\Delta$
  - $\beta_{64} = \gamma_{64} = 7\Delta$

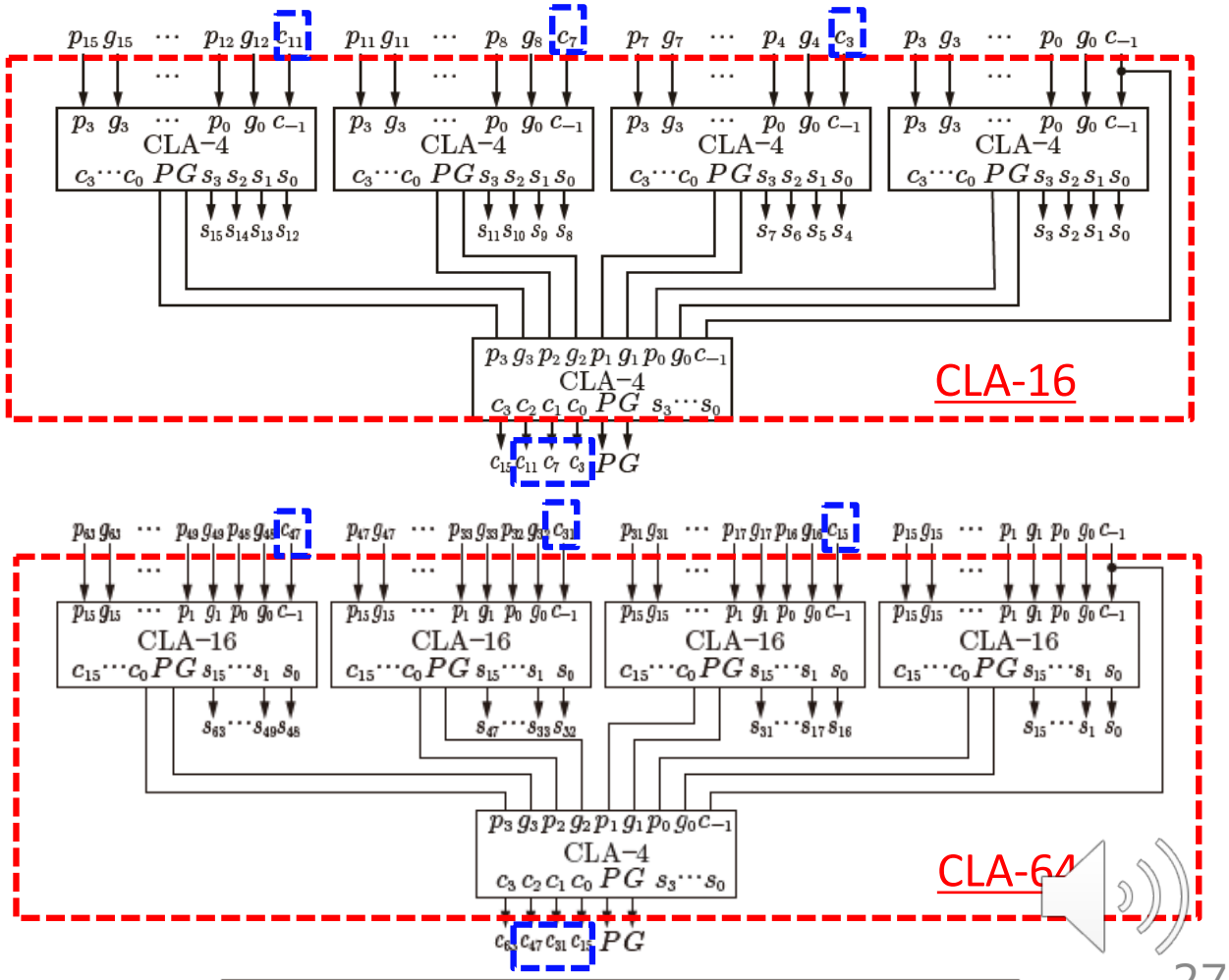


図 9・5　64 ビット桁上げ先見加算器の構成方法

６４ビット桁上げ先見加算器の遅延時間

64ビット桁上げ先見加算器（CLA-64）

- {CLA-16}４個＋CLA-4の２段回路で実現（・, ∨, ¬, ⊕ の遅延時間：⊿）
  - $\alpha_{4m} = \beta_m + 2\Delta + \gamma_m$
  - $\beta_{4m} = \beta_m + 2\Delta$
  - $\gamma_{4m} = \gamma_m + 2\Delta$
  - $\alpha_4 = 4\Delta$
  - $\beta_4 = \gamma_4 = 3\Delta$
  - $\alpha_{16} = 8\Delta$
  - $\beta_{16} = \gamma_{16} = 5\Delta$
  - $\alpha_{64} = 12\Delta$
  - $\beta_{64} = \gamma_{64} = 7\Delta$
  - $\alpha_4^k = 4k^*\Delta$
  - $\beta_4^k = (2k+1)^*\Delta$
  - $\gamma_4^k = (2k+1)^*\Delta$

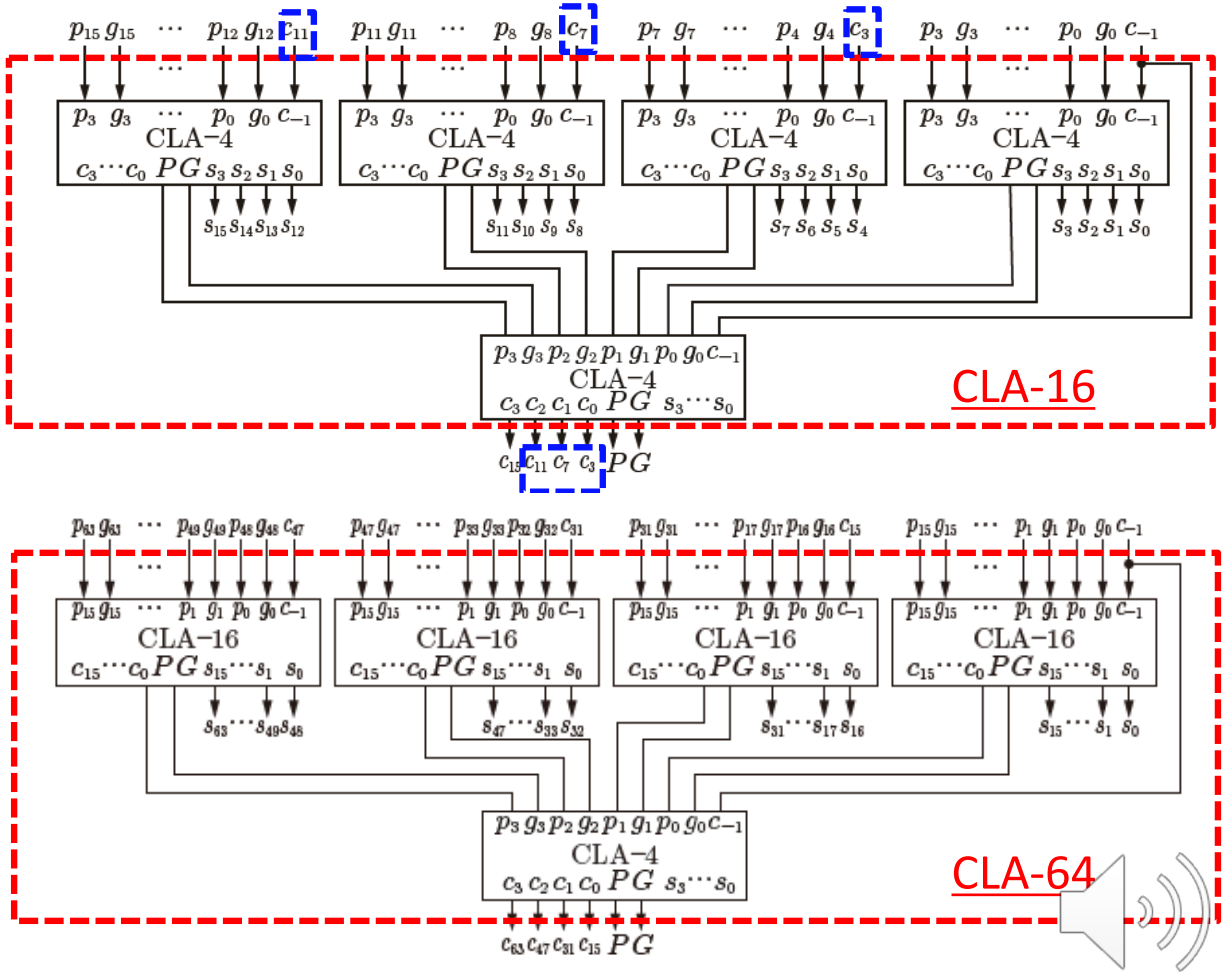


図 9・5　64 ビット桁上げ先見加算器の構成方法

4^kビット桁上げ先見加算器の遅延時間

4^kビット桁上げ先見加算器（CLA-4^K）

- {CLA-16}４個＋CLA-4の２段回路で実現（・, ∨, ¬, ⊕ の遅延時間：⊿）
  - $\alpha_{4^k} = 4k*\Delta$
  - $\beta_{4^k} = (2k+1)*\Delta$
  - $\gamma_{4^k} = (2k+1)*\Delta$
  - $n=4^K$ とすると
  - $\alpha_n = 4*(\log_4 n)*\Delta$
  - $\beta_n = (2*(\log_4 n)+1)*\Delta$
  - $\gamma_n = (2*(\log_4 n)+1)*\Delta$
- $O(\alpha_n)=O(\log_2 n)$
- “n” が大きくなると効率的！

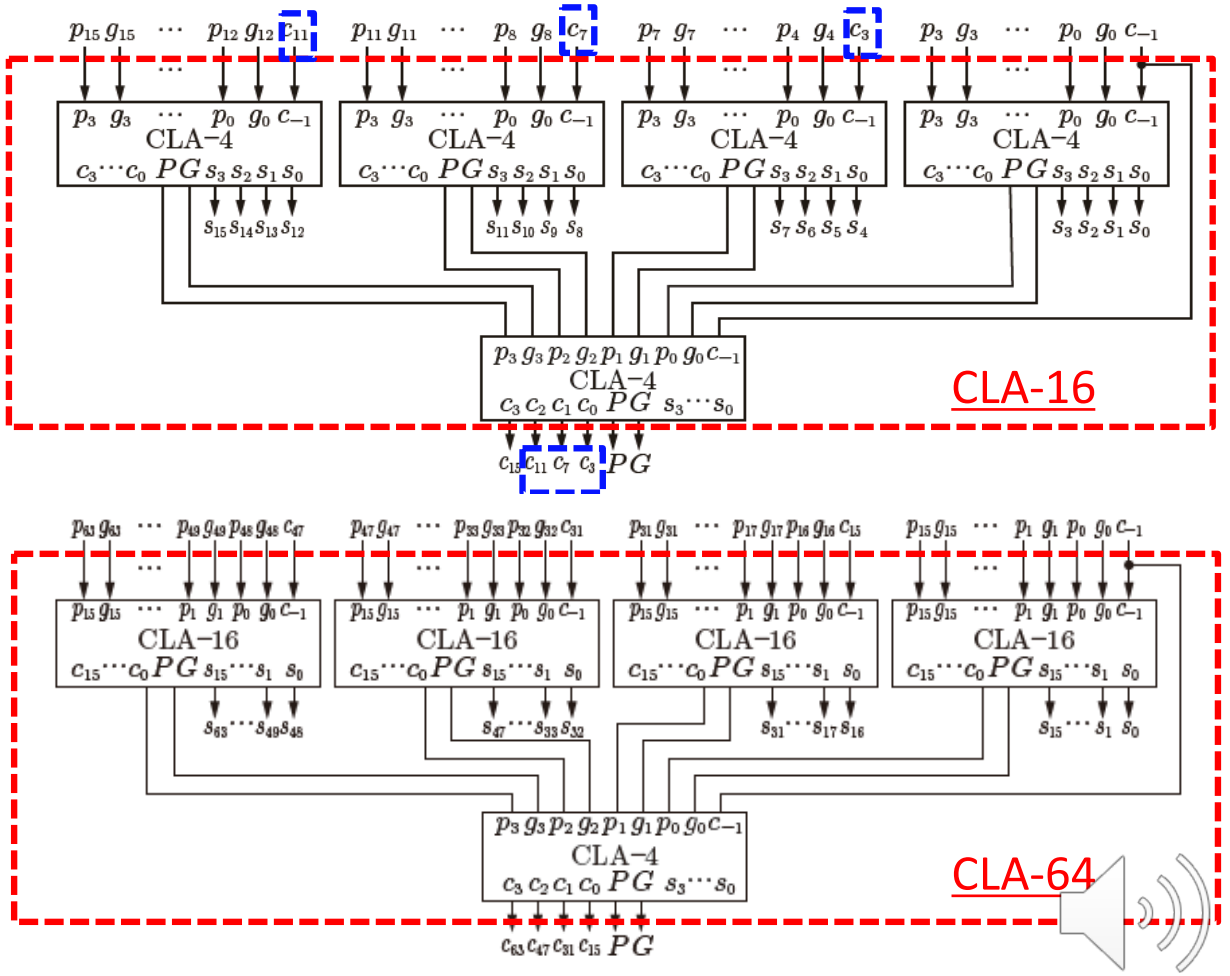


図 9・5　64 ビット桁上げ先見加算器の構成方法

逐次桁上げ加算器と桁上げ先見加算器の遅延時間の比較

- 1, 4, 16, 64 ビットの逐次桁上げ加算器と桁上げ先見加算器の遅延時間を 図9·6 に示す．この図からも，両加算器の遅延時間がそれぞれ O(n) および $O(\log_2 n)$ であることが確認できる．
- 4 ビット，16 ビット，64 ビットの桁上げ先見加算器は，同じ桁数の逐次桁上げ加算器と比較して，それぞれ 1.5 倍，3.1 倍，8.4 倍高速である．

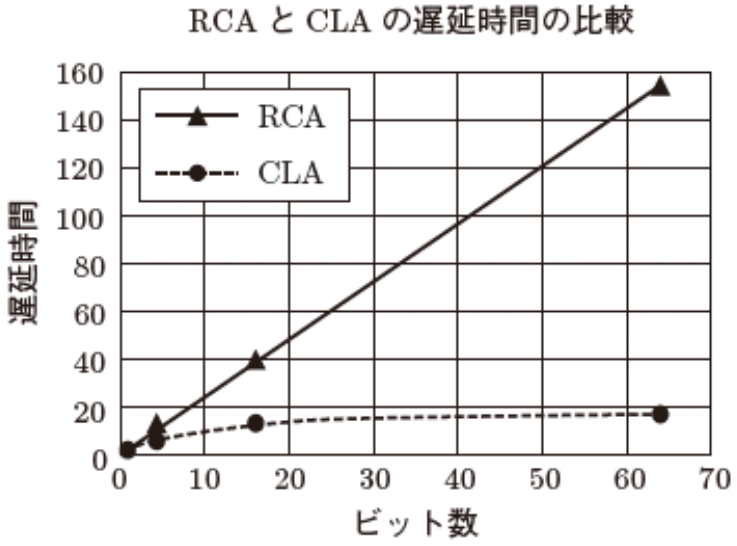


図 9・6　逐次桁上げ加算器と桁上げ先見加算器の遅延時間の比較

2 の補数と 1 の補数

- 2 の補数 は，n 桁の 2 進数を考えた場合，値を表現している数が桁上がりを生じるのに必要な最小の数である.
  - 4 ビットで表現すると 16 通りの数値を表現できるが，最上位ビット（MSB）を符号ビットとして用いる補数表現では 8 以上の数値の表現は行わずに −8 から 7 までの 16 通りの数値を表現している.
  - MSB が 0 のときは正の数，1 のときは負の数を表現している.
- 1 の補数 は n 桁の 2 進数で表現できる最大値（$2^n - 1$）を基準とし，その値から X を引いて負の数を表す補数表現である.
  - 1 の補数も 表1·6 に示されている.
  - 1 の補数では変換したい値 X のすべてのビットを 0 から 1 もしくは 1 から 0 に反転させることで，簡単に求めることができる.

表 1·6　2 の補数と 1 の補数

10 進表現	2 の補数	1 の補数
7	0111	0111
6	0110	0110
5	0101	0101
4	0100	0100
3	0011	0011
2	0010	0010
1	0001	0001
0	0000	0000(0),1111(−0)
−1	1111	1110
−2	1110	1101
−3	1101	1100
−4	1100	1011
−5	1011	1010
−6	1010	1001
−7	1001	1000
−8	1000	表現でき

2 の補数の求め方

- 2 の補数は，n ビットで表現された元の数を X とすると，$2^n - X$ で求められる．MSB を 1 とし，その後ろに n 桁の 0 が続いた数値から値 X を引くことになるため，ちょうど X の n 桁の 0, 1 を反転させ，その値に 1 を加算した場合と同じになる．この様子を 図1·10 に示す．
- 2 の補数の補数を計算すると，補数をとる前の元の数に戻る．
- n ビットで表現された 2 の補数 $X = (b_{n-1}, \ldots, b_1, b_0)$ の 10 進数への変換は，式 (1·7) で求めることができる．

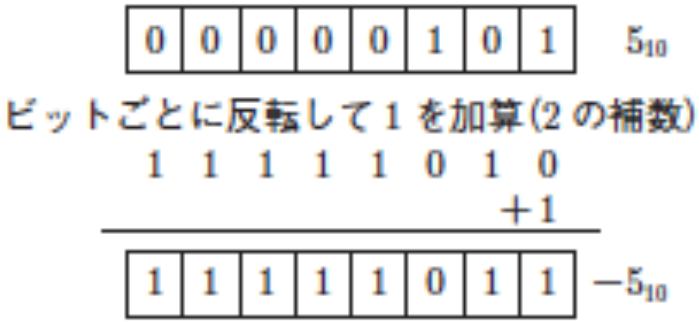


図 1·10 2 の補数の求め方

$$X = -b_{n-1} \cdot 2^{n-1} + b_{n-2} \cdot 2^{n-2} + \cdots + b_1 \cdot 2^1 + b_0 \cdot 2^0 \qquad (1 \cdot 7)$$

加減算器

- 加減算器は，加算と減算を選択的に実行できる演算器である．2 の補数形式の二つの 2 進数を x および y とする．x と y の加算は，加算器を用いて実行可能である．x から y を減ずる場合には，y の 2 の補数を x に加えればよい．
- y の 2 の補数は，y の 1 の補数（y のすべてのビットを反転させたもの）に 1 を加えることで得られる．

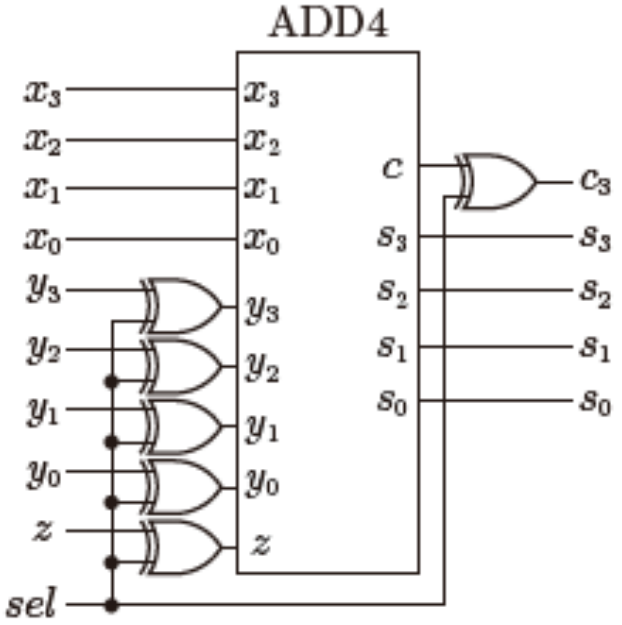


図 9・7　4 ビット加減算器

加減算器

- 選択信号を sel とし，加算を行う場合には sel = 0，減算を行う場合には sel = 1とする.
- 加算器の第二データ入力端子にはsel = 0 の場合には y の値をそのまま入力し，sel = 1 の場合には y の否定を入力するために加算器の第二データ入力端子に排他的論理和ゲートを接続し，sel と y の排他的論理和を入力する.
- また，桁上げ入力端子にも排他的論理和ゲートを接続し，下位桁からの桁上げ/桁借り信号と sel との排他的論理和を入力する．このようにして実装した桁上げ/桁借り入力付 4 ビット加減算器の回路図を図9·7 に示す.

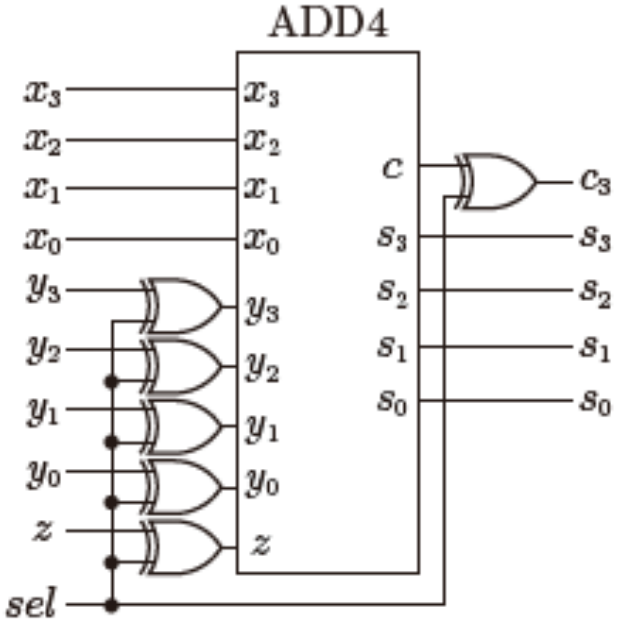


図 9・7　4 ビット加減算器

加減算器

- この加減算器を複数接続して，より多い桁数の加減算を行うためには，加減算器の桁上げ/桁借り信号を工夫する必要がある．符号なし加算器で減算を 2 の補数の加算によって実行した場合，結果が非負の場合には桁上げ信号が 1 になり，結果が負の場合には桁上げ信号が 0 になってしまい，桁借り信号の意味が逆になってしまうからである．そこで，加算器の桁上げ出力と sel の排他的論理和を加減算器の桁上げ/桁借りとして出力すれば正しく動作する．

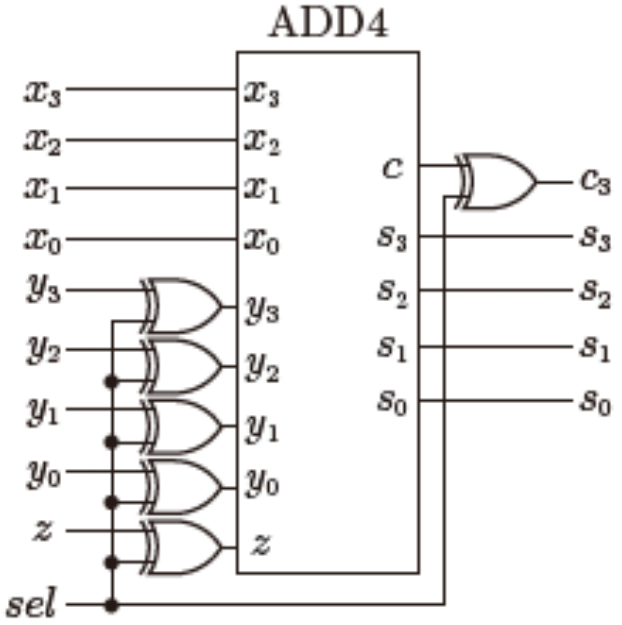


図 9・7　4 ビット加減算器

2 進化10 進加算器

- 10 進数を対象として数値演算を行う場合，10 進数の各桁を4 ビットの2 進数を用いた2 進化10 進数（binary coded decimal（BCD）number）で表現するのが一般的である.
- 2 進化 10 進加算器（BCD adder）は，二つの 2 進化 10 進数を入力し，2 進加算器を用いて加算を行ってから，必要であれば結果を 2 進化 10 進数に補正して演算結果を求める.
- すなわち，2 進加算器の出力が $(10)_{10}$ = $(1010)_2$ から $(15)_{10}$ = $(1111)_2$ までのいずれかの値になった場合，表9·1 の右側の 2 列に示すように加算結果を上位桁への桁上げとその桁の正しい 2 進化 10 進表現に分離・補正することで正しい結果を出力している.

表 9・1　2 進化 10 進加算での出力の補正

加算器の演算結果			補正後の出力	
桁上げ	和	10 進数	桁上げ	補正後の値
0	0000	0	0	0000
0	0001	1	0	0001
0	0010	2	0	0010
0	0011	3	0	0011
0	0100	4	0	0100
0	0101	5	0	0101
0	0110	6	0	0110
0	0111	7	0	0111
0	1000	8	0	1000
0	1001	9	0	1001
0	1010	10	1	0000
0	1011	11	1	0001
0	1100	12	1	0010
0	1101	13	1	0011
0	1110	14	1	0100
0	1111	15	1	0101
1	0000	16	1	0110
1	0001	17	1	0111
1	0010	18	1	000
1	0011	19	1	001

2 進化10 進加算器

- 2 進化 10 進数の加算器は次のようにして実現できる．まず，加算器の二つの入力はいずれも，0 から 9 の間の値および下位の桁からの桁上げ（0 または 1）であるとする．演算結果は 0 から 19 の間の値になる．この演算を 2 進数の加算器でそのまま実行した場合の演算結果は次のように分類できる．
  - 加算結果が 0 以上 9 以下の場合には，桁上げは発生しない．演算結果を修正する必要はない．
  - 加算結果が 10 以上 15 以下の場合には，桁上げは発生しない．演算結果の補正を行う必要がある．
  - 演算結果が 16 以上 19 以下の場合には，桁上げが発生する．演算結果の補正を行う必要がある．

表 9・1　2 進化 10 進加算での出力の補正

加算器の演算結果			補正後の出力	
桁上げ	和	10 進数	桁上げ	補正後の値
0	0000	0	0	0000
0	0001	1	0	0001
0	0010	2	0	0010
0	0011	3	0	0011
0	0100	4	0	0100
0	0101	5	0	0101
0	0110	6	0	0110
0	0111	7	0	0111
0	1000	8	0	1000
0	1001	9	0	1001
0	1010	10	1	0000
0	1011	11	1	0001
0	1100	12	1	0010
0	1101	13	1	0011
0	1110	14	1	0100
0	1111	15	1	0101
1	0000	16	1	0110
1	0001	17	1	0111
1	0010	18	1	1000
1	0011	19	1	1001

2 進化10 進加算器

- したがって，2 進数の加算器の出力が 10 以上 19 以下の場合には桁上げを行い，2 進数の加算器の出力に対して 10 進補正（decimal adjustment）と呼ばれる補正を行う必要がある．補正は加算器の出力に 6 を加えるだけでよい．
- まず，加算結果が 10 以上 15 以下の場合には，加算器からの桁上げはない．この出力に 6 を加えると，結果が16 以上21 以下になるので桁上げが発生する．桁上げを除く加算結果は 0 から 5 のいずれかになる．上位桁への桁上げは 10 として扱うので，期待どおりの結果（10 から15）が得られることになる．

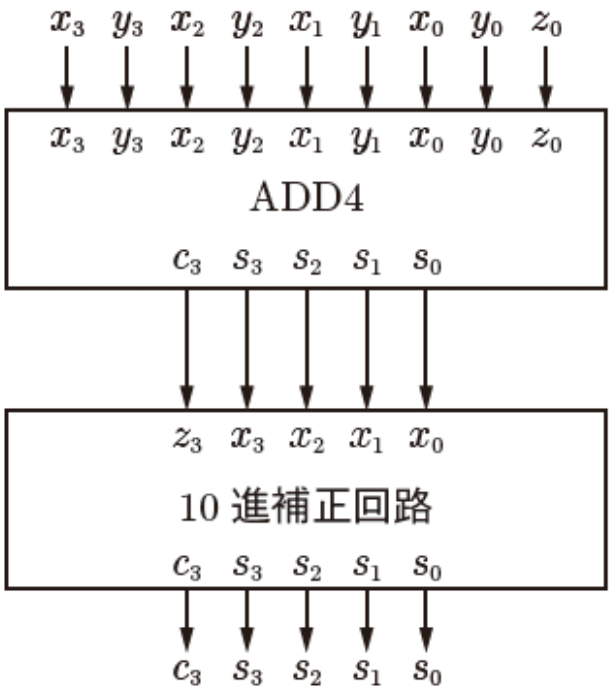


図 9・8　2 進化 10 進加算器

2 進化10 進加算器

- 次に，加算結果が 16 以上の場合には，加算器からの桁上げ出力があり，桁上げを除く出力は，0 から 3 の値になる．この出力に 6 を加えると，結果は 6 から 9 の値になり，新たな桁上げは発生しない．最初の桁上げは 10 として扱うので，期待する結果（16 から19）が得られることになる．
- 上記の考察をまとめると次のようになる．最初の加算の結果が 10 を超えた場合に，桁上げを発生させ，かつ加算器の出力に 6 を加えれば期待どおりの結果が得られる．ただし，6 を加えたときに桁上げが発生してもこの桁上げは無視する．
- このようにして設計された 2 進化10 進加算器の構成を 図9·8 に示す．

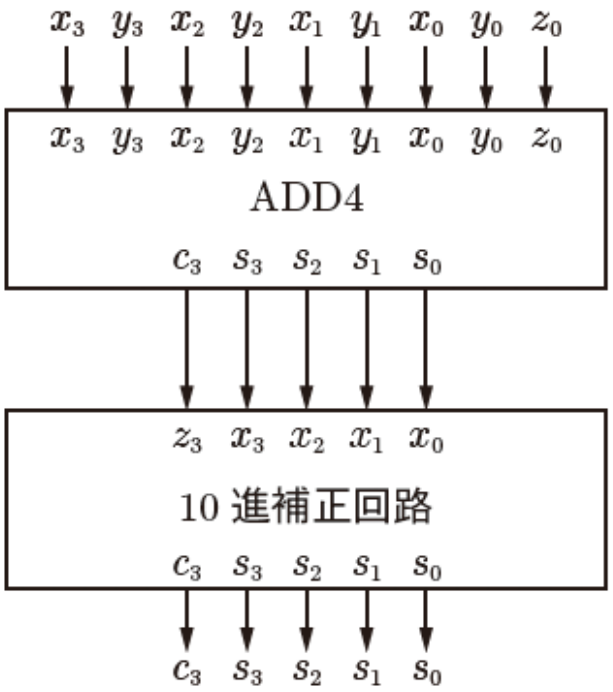


図 9・8　2 進化 10 進加算器

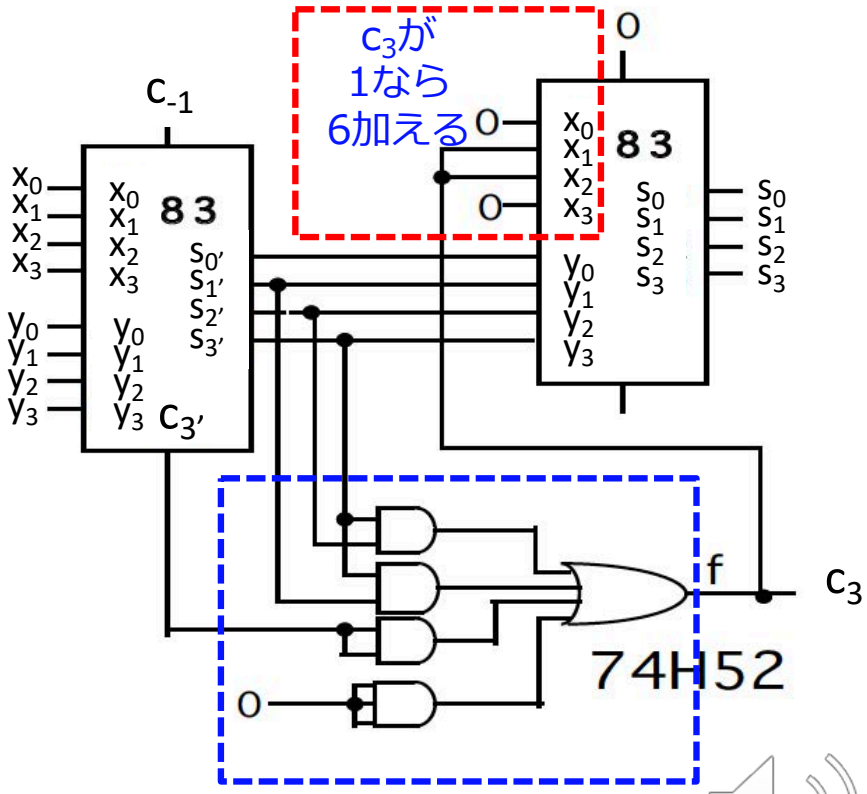
10進補正回路

10進補正回路は次のように構成できる

- $X=(x_3,x_2,x_1,x_0)$, $Y=(y_3,y_2,y_1,y_0)$ に対して，2進化10進の和 $S=(s_3,s_2,s_1,s_0)$と上位への桁上げ c_3 を求める際の10進補正回路は，次のように実現できる.
  - 上位の桁上げは X+Y が 10 以上のとき $c_3=1$
  - X+Y が 10 以上のときは $S=X+Y+6$
  - X+Y が 16 以上のとき $c_{3'}=1$ になり，X+Y が 10～15 のとき $s_{3'}\cdot s_{2'} \vee s_{3'}\cdot s_{1'}=1$ になるので，

$$c_3 = c_{3'} \vee s_{3'}\cdot s_{2'} \vee s_{3'}\cdot s_{1'}$$

7400シリーズの集積回路を使って次のように構成できる（7483,74H52を使った構成）



算術論理ユニット（ALU）

- 算術論理ユニット（Arithmetic and Logical Unit：ALU）
  - ALU は 2 進数の加減算などの算術演算の他に，ビットごとの論理演算（∨，・，⊕，¬）も実行できる，汎用性の高い演算器
- 全加算器の y_i 入力に論理ゲートを追加して，図9·9 の加減算器を構成する．この回路を必要なビット数分用意し，下位桁の桁上げ出力を上位桁の桁上げ入力に接続する．このようにして構成された演算器は，表9·2 に示す機能をもつ．

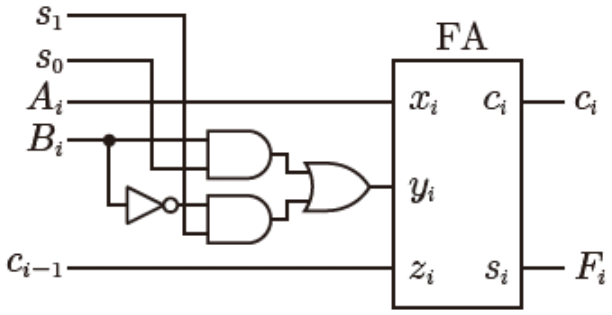


図 9・9　ALU で用いる加減算器

表 9・2　ALU の算術演算機能

s_1	s_0	c_{-1}	Y	出力	機能
0	0	0	0	A	転送
0	0	1	0	$A+1$	インクリメント
0	1	0	B	$A+B$	加算
0	1	1	B	$A+B+1$	桁上げ付き加算
1	0	0	$\overline{B}$	$A-B-1$	桁借り付き減算
1	0	1	$\overline{B}$	$A-B$	減算
1	1	0	1	$A-1$	ディクリメ
1	1	1	1	A	転送

OSAKA UNIVERSITY

演習問題

1 図 9·1（c）の全加算器が図 9·1（a）の全加算器と論理的に等価であることを示せ．

2 全減算器を設計し，その論理回路図を示せ．全減算器の入力は，被減数 x，減数 y，下位の桁からの借り上げ z とする．また出力は，借り上げ b，差 d とする．

3 4 ビットの符号なしインクリメンタ（incrementer）を設計せよ．インクリメンタの入力は x とし，出力は桁上げ c および y とする．このインクリメンタでは，$0 \leq x$ かつ $x < 14$ の場合には，$c = 0$，$y = x + 1$ が出力される．また，$x = 15$ の場合には，$c = 1$，$y = 0$ が出力される．

4 4 ビットの符号なしディクリメンタ（decrementer）を設計せよ．ディクリメンタの入力は x とし，出力は桁借り b および y とする．このディクリメンタでは，$0 < x$ かつ $x \leq 15$ の場合には，$c = 0$，$y = x - 1$ が出力される．また，$x = 0$ の場合には，$b = 1$，$y = 15$ が出力される．

5 本章の 9·7 節で説明した ALU に，次のフラグを追加せよ．各フラグを生成するための論理式を示すこと．

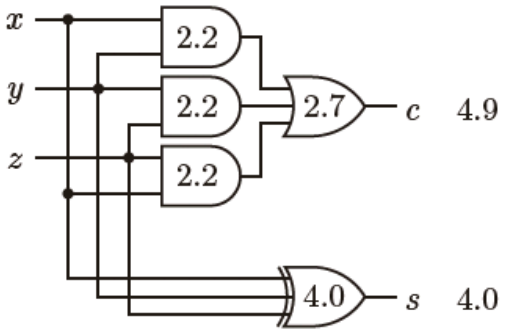
C：Carry（桁上げ，減算時は借り上げ）
Z：Zero（加減算の結果が 0）
S：Sign（符号ビット，加減算の結果の最上位桁）
V：Overflow（桁あふれ）
P: Parity（演算結果の全ビットの排他的論理和）

6 図 9·3 の 4 ビット桁上げ先見加算器の各出力信号の遅延時間を求めよ．

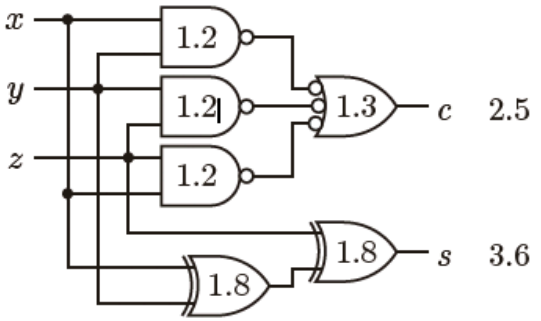
7 図 9·3 の 16 ビット桁上げ先見加算器の桁上げ信号 c_{15} および和信号 s_{15} の遅延時間をそれぞれ求めよ．

演習問題 1

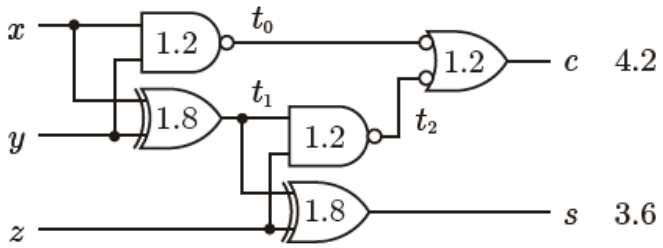
- 図9·1(c) の全加算器が図9·1(a) の全加算器と論理的に等価であることを示せ



（a） AND, OR, XOR ゲートを用いた実装



（b） NAND, XOR ゲートを用いた実装(1)



（c） NAND, XOR ゲートを用いた実装(2)

図 9・1 全加算器

考慮時間

- 5分間程度で問題を解いてみてください．その間，ビデオを止めてください.

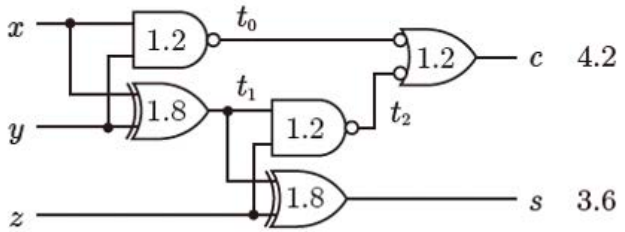
- この頁は30秒程度で次の頁に移行します.

演習問題１解答

- 図9·1(c) の全加算器が図9·1(a) の全加算器と論理的に等価であることを示せ.

（解答）

- 図9·1（c）の出力信号c および s の論理式を求め，図9·1(a) の出力信号c および s の論理式と比較すると一致する.
  - $t_0 = \overline{xy} = \bar{x} \vee \bar{y}$
  - $t_1 = x \oplus y = \bar{x}y \vee x\bar{y}$
  - $t_2 = \overline{t_1} \vee \bar{z} = \neg(\bar{x}y \vee x\bar{y}) \vee \bar{z} = \cdots = xy \vee \bar{x}\bar{y} \vee \bar{z}$
  - $c = \overline{t_0} \vee \overline{t_2} = xy \vee \neg(xy \vee \bar{x}\bar{y} \vee \bar{z}) = \cdots = xy + x\bar{y}z + \bar{x}yz$
  - $s = t_1 \oplus z = \overline{t_1}z \vee t_1\bar{z} = \cdots = xyz \vee x\bar{y}\bar{z} \vee \bar{x}y\bar{z} + \bar{x}\bar{y}z$



（c） NAND, XOR ゲートを用いた実装(2)

第12章
順序回路の簡単化と順序回路の例

第12章　順序回路の簡単化と順序回路の例

- この章のねらい

12章 順序回路の簡単化と順序回路の例

本章では，前章で説明した順序回路設計時の注意点と実際の回路例を紹介する．まず，順序回路の簡単化について，状態数最小化および状態割当てについて説明する．その後，Mealy 型順序機械と Moore 型順序機械の変換法について説明する．また，実際の順序回路の例として，6 進カウンタ，フィルタ回路，シフトレジスタ，シリアル・パラレル変換回路を紹介する．

順序回路の簡単化

- 状態数最小化
  - 状態数は，順序回路の規模（ハードウェア量，面積）に大きな影響を与えるため，できるだけ小さいことが望ましい．前章で，状態の割当てによって，順序回路の規模は影響を受けることを述べたが，同様に状態数も重要な要素である．したがって，最初に，状態遷移図，状態遷移表を作成したときに，その状態数が最小になっているかを検討することは重要である．
  - すでに作成した状態遷移図，状態遷移表から状態数を削減するためには，共用できうる状態はまとめて一つの状態として扱う．共用できる状態とは，同一入力に対して同一の出力を行い，かつ同一の状態に遷移する状態である．言い換えると，外部から見たときに区別のつかない状態である．この場合，共用できる状態は，単一の状態として置き換えても矛盾は生じない．この操作を状態の縮退という．状態の縮退を行い，状態数を最小化することを状態数最小化という．

状態数最小化の手順

- 状態数最小化の手順
  1. 同一入力に対して同一出力となる状態をグループ化
  2. 新たに決定したグループを使って，状態ごとに状態遷移先を更新
  3. 同一グループからの遷移先が唯一になるようにグループを再分割
  4. 分割の更新がなくなるまで，2.，3. を繰り返す

（例題）

- 表１２・１の状態数最小化？

表 12・1　状態遷移出力表の例

現在の状態	次の状態		出力	
	x_0	x_1	x_0	x_1
S_0	S_0	S_1	z_0	z_0
S_1	S_3	S_5	z_1	z_0
S_2	S_2	S_1	z_0	z_0
S_3	S_4	S_5	z_0	z_0
S_4	S_2	S_3	z_1	z_1
S_5	S_4	S_0	z_1	z_0

状態数最小化の手順

- 状態数最小化の手順
  1. 同一入力に対して同一出力となる状態をグループ化
  2. 新たに決定したグループを使って，状態ごとに状態遷移先を更新
  3. 同一グループからの遷移先が唯一になるようにグループを再分割
  4. 分割の更新がなくなるまで，2.，3. を繰り返す

（例題）

表 12・1　状態遷移出力表の例

現在の状態	次の状態		出力	
	x_0	x_1	x_0	x_1
S_0	S_0	S_1	z_0	z_0
S_1	S_3	S_5	z_1	z_0
S_2	S_2	S_1	z_0	z_0
S_3	S_4	S_5	z_0	z_0
S_4	S_2	S_3	z_1	z_1
S_5	S_4	S_0	z_1	z_0

表 12・2　グループ化 1

グループ	現在の状態	次の状態		出力		遷移先グループ	
		x_0	x_1	x_0	x_1	x_0	x_1
A	S_0	S_0	S_1	z_0	z_0	A	B
	S_2	S_2	S_1	z_0	z_0	A	B
	S_3	S_4	S_5	z_0	z_0	C	B
B	S_1	S_3	S_5	z_1	z_0	A	B
	S_5	S_4	S_0	z_1	z_0	C	A
C	S_4	S_2	S_3	z_1	z_1	A	A

状態数最小化の手順

- 状態数最小化の手順
  1. 同一入力に対して同一出力となる状態をグループ化
  2. 新たに決定したグループを使って，状態ごとに状態遷移先を更新
  3. 同一グループからの遷移先が唯一になるようにグループを再分割
  4. 分割の更新がなくなるまで，2.，3. を繰り返す

（例題）

表 12・2　グループ化 1

グループ	現在の状態	次の状態		出力		遷移先グループ	
		x_0	x_1	x_0	x_1	x_0	x_1
A	S_0	S_0	S_1	z_0	z_0	A	B
	S_2	S_2	S_1	z_0	z_0	A	B
	S_3	S_4	S_5	z_0	z_0	C	B
B	S_1	S_3	S_5	z_1	z_0	A	B
	S_5	S_4	S_0	z_1	z_0	C	A
C	S_4	S_2	S_3	z_1	z_1	A	A

表 12・3　グループ化 2

グループ	現在の状態	次の状態		出力		遷移先グループ	
		x_0	x_1	x_0	x_1	x_0	x_1
A_1	S_0	S_0	S_1	z_0	z_0	A	B
	S_2	S_2	S_1	z_0	z_0	A	B
A_2	S_3	S_4	S_5	z_0	z_0	C	B
B_1	S_1	S_3	S_5	z_1	z_0	A	B
B_2	S_5	S_4	S_0	z_1	z_0	C	
C	S_4	S_2	S_3	z_1	z_1	A	

状態数最小化の手順

- 状態数最小化の手順
  1. 同一入力に対して同一出力となる状態をグループ化
  2. 新たに決定したグループを使って，状態ごとに状態遷移先を更新
  3. 同一グループからの遷移先が唯一になるようにグループを再分割
  4. 分割の更新がなくなるまで，2.，3. を繰り返す

（例題）

表 12・3　グループ化 2

グループ	現在の状態	次の状態		出力		遷移先グループ	
		x_0	x_1	x_0	x_1	x_0	x_1
A_1	S_0	S_0	S_1	z_0	z_0	A	B
	S_2	S_2	S_1	z_0	z_0	A	B
A_2	S_3	S_4	S_5	z_0	z_0	C	B
B_1	S_1	S_3	S_5	z_1	z_0	A	B
B_2	S_5	S_4	S_0	z_1	z_0	C	A
C	S_4	S_2	S_3	z_1	z_1	A	A

表 12・4　グループ化 3

グループ	現在の状態	次の状態		出力		遷移先グループ	
		x_0	x_1	x_0	x_1	x_0	x_1
A_1	S_0	S_0	S_1	z_0	z_0	A_1	B_1
	S_2	S_2	S_1	z_0	z_0	A_1	B_1
A_2	S_3	S_4	S_5	z_0	z_0	C	B_2
B_1	S_1	S_3	S_5	z_1	z_0	A_2	B_2
B_2	S_5	S_4	S_0	z_1	z_0	C	A_1
C	S_4	S_2	S_3	z_1	z_1	A_1	A_2

状態数最小化の手順

- 状態数最小化の手順
  1. 同一入力に対して同一出力となる状態をグループ化
  2. 新たに決定したグループを使って，状態ごとに状態遷移先を更新
  3. 同一グループからの遷移先が唯一になるようにグループを再分割
  4. 分割の更新がなくなるまで，2.，3. を繰り返す

（例題）

状態S_0,S_2は区別がつかない状態

表 12・1　状態遷移出力表の例

現在の状態	次の状態		出力	
	x_0	x_1	x_0	x_1
S_0	S_0	S_1	z_0	z_0
S_1	S_3	S_5	z_1	z_0
S_2	S_2	S_1	z_0	z_0
S_3	S_4	S_5	z_0	z_0
S_4	S_2	S_3	z_1	z_1
S_5	S_4	S_0	z_1	z_0

表 12・4　グループ化 3

グループ	現在の状態	次の状態		出力		遷移先グループ	
		x_0	x_1	x_0	x_1	x_0	x_1
A_1	S_0	S_0	S_1	z_0	z_0	A_1	B_1
	S_2	S_2	S_1	z_0	z_0	A_1	B_1
A_2	S_3	S_4	S_5	z_0	z_0	C	B_2
B_1	S_1	S_3	S_5	z_1	z_0	A_2	B_2
B_2	S_5	S_4	S_0	z_1	z_0	C	
C	S_4	S_2	S_3	z_1	z_1	A_1	A_2

状態数最小化の手順

- 状態数最小化の手順
  1. 同一入力に対して同一出力となる状態をグループ化
  2. 新たに決定したグループを使って，状態ごとに状態遷移先を更新
  3. 同一グループからの遷移先が唯一になるようにグループを再分割
  4. 分割の更新がなくなるまで，2.，3. を繰り返す

（例題）

状態S_0,S_2は区別がつかない状態

表 12・1　状態遷移出力表の例

現在の状態	次の状態		出力	
	x_0	x_1	x_0	x_1
S_0	S_0	S_1	z_0	z_0
S_1	S_3	S_5	z_1	z_0
S_2	S_2	S_1	z_0	z_0
S_3	S_4	S_5	z_0	z_0
S_4	S_2	S_3	z_1	z_1
S_5	S_4	S_0	z_1	z_0

表 12・5　最小化後の状態遷移

現在の状態	次の状態		出力	
	x_0	x_1	x_0	x_1
$S_{0,2}$	$S_{0,2}$	S_1	z_0	z_0
S_1	S_3	S_5	z_1	z_0
S_3	S_4	S_5	z_0	z_0
S_4	$S_{0,2}$	S_3	z_1	
S_5	S_4	$S_{0,2}$	z_1	z_0

状態割当て

- 状態の割当てによっては，回路を簡単化できる場合がある．近年はコンピュータ支援設計（CAD: Computer Aided Design）技術の進歩により，適切な状態割当てを自動的に行えるようになってきているが，状態の割当てにより，実現する回路に違いができることを理解しておくことは重要である．

例題12・2

- 表12･6 に示すように状態を割り当てたときの“110” 文字検出回路を設計せよ．

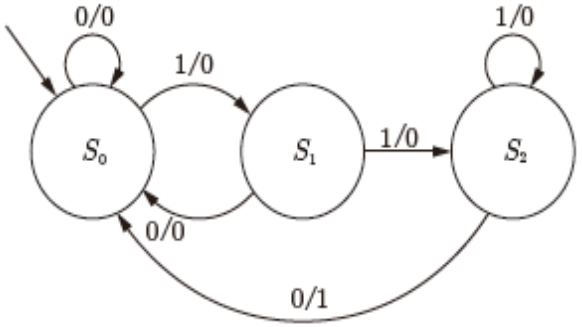


図 11・5　“110” 文字列検出回路（Mealy 型）

表 11・3　“110” 文字列検出回路の状態遷移出力表（Mealy 型）

現在の状態	次の状態		出力	
	$x=0$	$x=1$	$x=0$	$x=1$
S_0	S_0	S_1	0	0
S_1	S_0	S_2	0	0
S_2	S_0	S_2	1	0

表 12・6　状態割当て

状態	表 11･5 の状態割当て	別の状態割当て
S_0	00	00
S_1	01	01
S_2	11	10

3.状態遷移関数，出力関数の実現

- Mealy型順序機械の状態割当て決定後の状態遷移出力表を表11･5 に示す.
- ここで，状態を表すための変数 q_1, q_0 を導入した．次の時刻の状態を表現する状態変数を q_1^+, q_0^+ と表すと，状態変数q_1^+, q_0^+ および出力 z は，現在時刻の状態変数 q_1, q_0 および入力 x を使った論理関数として表現できる．状態遷移出力表から， q_1^+, q_0^+, z に関するカルノー図を作成すると図11･6 のカルノー図が得られる．10 は状態割当てされていないため， q_1, q_0 の論理式を決める際にはドントケアの組合せとなり，図11･6 中では，‘X’ と表されている.

表 11・5　状態割当て決定後の状態遷移出力表（Mealy 型）

現在の状態	次の状態		出力	
$q_1 q_0$	$x=0$	$x=1$	$x=0$	$x=1$
00	00	01	0	0
01	00	11	0	0
11	00	11	1	0

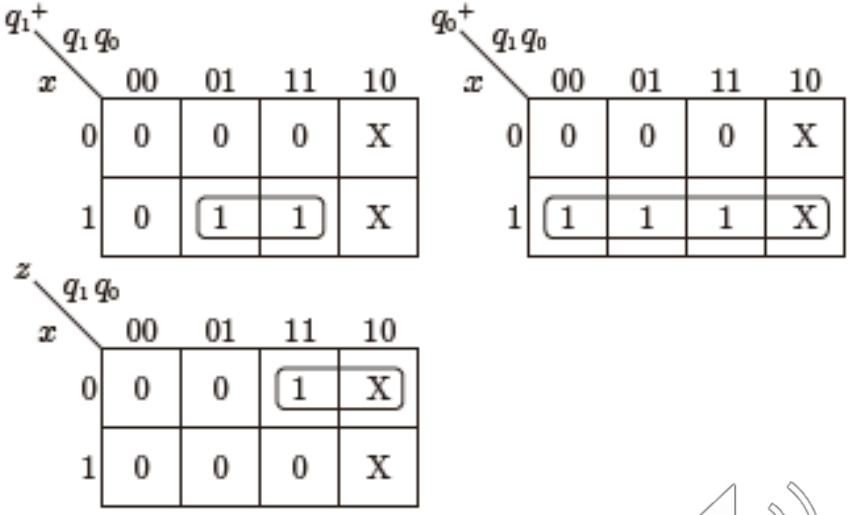


図 11・6　カルノー図（Mealy 型）

3.状態遷移関数，出力関数の実現

- Mealy型順序機械の状態割当て決定後の状態遷移出力表を表11･5 に示す.
- ここで，状態を表すための変数 q_1, q_0 を導入した．次の時刻の状態を表現する状態変数を q_1^+, q_0^+ と表すと，状態変数q_1^+, q_0^+ および出力 z は，現在時刻の状態変数 q_1, q_0 および入力 x を使った論理関数として表現できる．状態遷移出力表から， q_1^+, q_0^+, z に関するカルノー図を作成すると図11･6 のカルノー図が得られる．10 は状態割当てされていないため， q_1, q_0 の論理式を決める際にはドントケアの組合せとなり，図11･6 中では，‘X’ と表されている.
- 図11･6 のカルノー図より，状態遷移関数，出力関数を求めると

$$q_1^+ = x \cdot q_0$$
$$q_0^+ = x$$
$$z = \bar{x} \cdot q_1$$

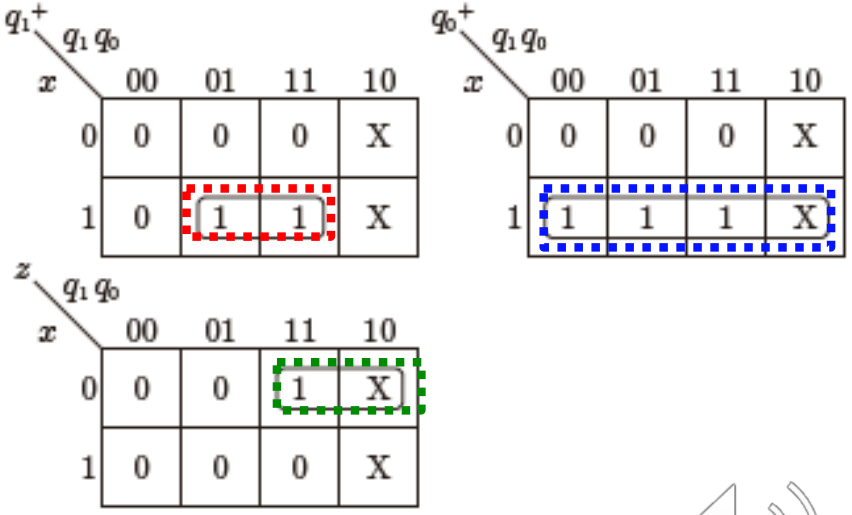


図 11・6 カルノー図（Mealy 型）

論理ゲートを使った順序回路の実現

- 状態遷移関数，出力関数が求まっているので，それらを実現する論理回路を構成する．記憶要素としては，状態変数の個数分の Dフリップフロップを用いる．
- 本文字列検出回路では，状態保持するために 2ビットの変数を使用するので，Dフリップフロップを 2 個使用する．
- 初期状態は動作開始前に Dフリップフロップのリセット端子 RST に ‘1’ の信号を与え，Dフリップフロップを初期化し，状態を “00” としている．
- Mealy 型順序回路で実現した回路構成を図11·7に示す．

$$q_1^+ = x \cdot q_0$$
$$q_0^+ = x$$
$$z = \bar{x} \cdot q_1$$

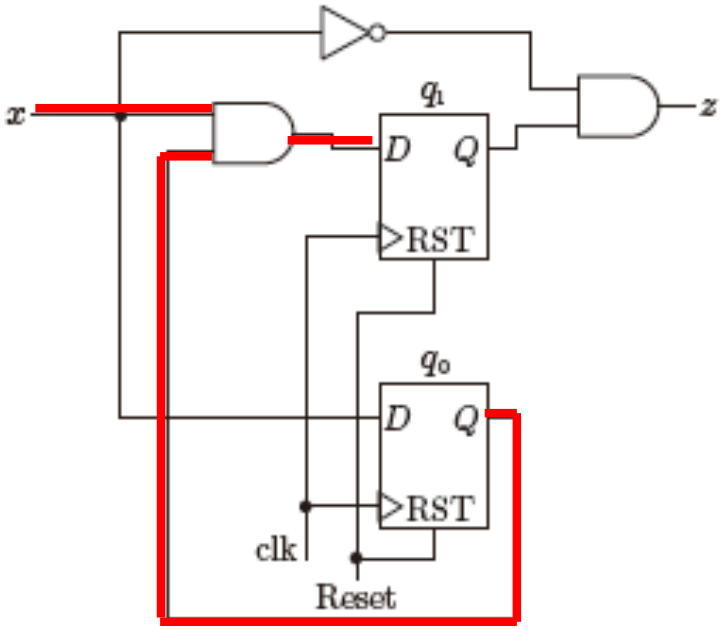


図 11・7　順序回路（Mealy 型）

状態割当て

- 状態の割当てによっては，回路を簡単化できる場合がある．近年はコンピュータ支援設計（CAD: Computer Aided Design）技術の進歩により，適切な状態割当てを自動的に行えるようになってきているが，状態の割当てにより，実現する回路に違いができることを理解しておくことは重要である．

例題12・2

- 表12･6 に示すように状態を割り当てたときの“110” 文字検出回路を設計せよ．

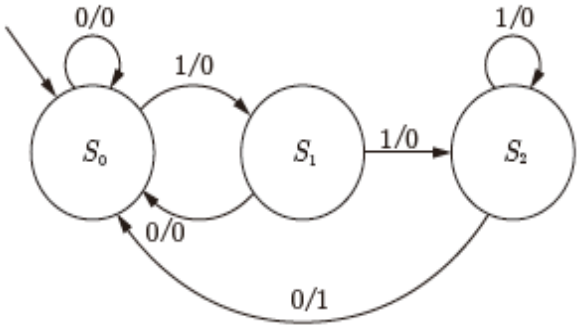


図 11・5 “110” 文字列検出回路（Mealy 型）

表 11・3 “110” 文字列検出回路の状態遷移出力表（Mealy 型）

現在の状態	次の状態		出力	
	$x=0$	$x=1$	$x=0$	$x=1$
S_0	S_0	S_1	0	0
S_1	S_0	S_2	0	0
S_2	S_0	S_2	1	0

表 12・6 状態割当て

状態	表 11･5 の状態割当て	別の状態割当て
S_0	00	00
S_1	01	01
S_2	11	10

状態遷移関数，出力関数の実現

- Mealy型順序機械の状態割当て決定後の状態遷移出力表を表11･5 に示す.
- ここで，状態を表すための変数 q_1, q_0 を導入した．次の時刻の状態を表現する状態変数を q_1^+, q_0^+ と表すと，状態変数q_1^+, q_0^+ および出力 z は，現在時刻の状態変数 q_1, q_0 および入力 x を使った論理関数として表現できる．状態遷移出力表から， q_1^+, q_0^+, z に関するカルノー図を作成するとき、11 は状態割当てされていないため， q_1, q_0 の論理式を決める際にはドントケアの組合せとなる.

表 12・6　状態割当て

状態	表 11･5 の状態割当て	別の状態割当て
S_0	00	00
S_1	01	01
S_2	11	10

表 12・7　状態割当決定後の状態遷移出力表（Mealy 型）

現在の状態	次の状態		出力	
$q_1 q_0$	$x=0$	$x=1$	$x=0$	$x=1$
00	00	01	0	0
01	00	10	0	0
10	00	10	1	0

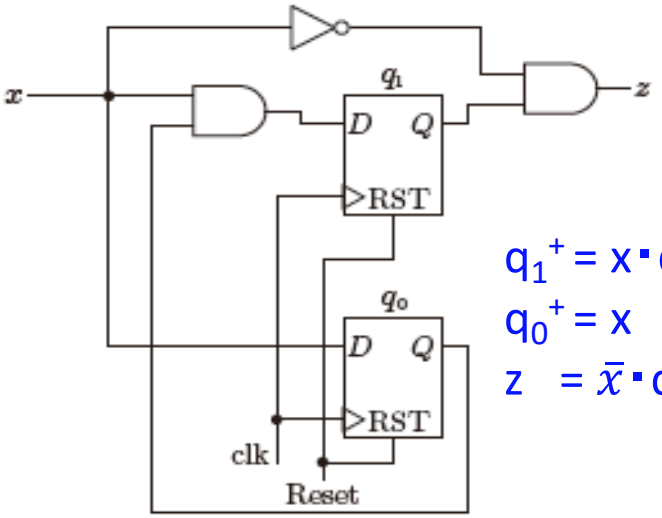
$q_1^+ = x \cdot q_0$　　$q_1^+ = x \cdot q_0 \vee x \cdot q_1$

$q_0^+ = x$　　$q_0^+ = q_1 \cdot q_0 \cdot x$

$z = \bar{x} \cdot q_1$　　$z = x \cdot q_1$

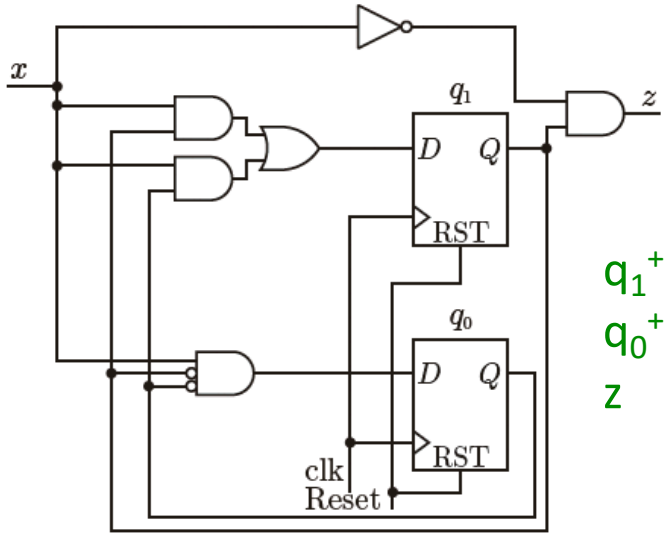
論理ゲートを使った順序回路の実現

- 図11·7 と図12·1 から，状態の割当てにより生成される回路が異なることがわかる．図11·7と図12·1 を比較すると，状態数が同じなので，両図とも 2 個の D フリップフロップが使われている．しかしながら，組合せ部分は，図11·7 は 2 個の 2 入力ANDゲートと 1 個のNOT ゲートから構成されるのに対して，図12·1 はかなり大規模な構成となり，回路規模から考えて図11·7の回路のほうが優れていることがわかる．以上より，状態の割当てにより生成される回路が大きく異なることがわかる．



$q_1^+ = x \cdot q_0$
$q_0^+ = x$
$z = \bar{x} \cdot q_1$

図 11・7　順序回路（Mealy 型）



$q_1^+ = x \cdot q_0 \vee x \cdot q_1$
$q_0^+ = q_1 \cdot q_0 \cdot x$
$z = x \cdot q_1$

図 12・1　順序回路（Mealy 型）別状態割当て版

演習問題

1 表 12·14 で与えられた状態遷移出力表がある．この表の状態数を最小化せよ．

表 12・14　ある順序回路の状態遷移出力表

現在の状態	次の状態		出力	
	x_0	x_1	x_0	x_1
S_0	S_1	S_7	z_1	z_1
S_1	S_0	S_7	z_0	z_0
S_2	S_2	S_0	z_0	z_0
S_3	S_3	S_0	z_0	z_0
S_4	S_1	S_4	z_1	z_1
S_5	S_5	S_0	z_0	z_0
S_6	S_1	S_6	z_1	z_1
S_7	S_0	S_1	z_1	z_0

2 図 12·4 の 2 進カウンタの状態数を最小化し，図 12·2 と一致することを確認せよ．

3 5 進カウンタを設計せよ．

4 12·3〔2〕項のフィルタ回路で，状態割当てを $S_0 = 00, S_1 = 01, S_2 = 11, S_3 = 10$ としたときの出力方程式，状態変数更新式をそれぞれ最簡積和形で求め，(12.7),(12.8),(12.9) と比較せよ．

OSAKA UNIVERSITY

演習問題 1

- 表12·14 で与えられた状態遷移出力表がある．この表の状態数を最小化せよ．

表 12・14　ある順序回路の状態遷移出力表

現在の状態	次の状態		出力	
	x_0	x_1	x_0	x_1
S_0	S_1	S_7	z_1	z_1
S_1	S_0	S_7	z_0	z_0
S_2	S_2	S_0	z_0	z_0
S_3	S_3	S_0	z_0	z_0
S_4	S_1	S_4	z_1	z_1
S_5	S_5	S_0	z_0	z_0
S_6	S_1	S_6	z_1	z_1
S_7	S_0	S_1	z_1	z_0

考慮時間

- 5分間程度で問題を解いてみてください．その間，ビデオを止めてください.

- この頁は30秒程度で次の頁に移行します.

演習問題１解答

- 表12·14 で与えられた状態遷移出力表がある．この表の状態数を最小化せよ．

（解答）

- 解表12·1 となる．従って，8 状態から 5 状態に最小化できる．

解表 12・1　グループ化後

グループ	現在の状態	次の状態		出力		遷移先グループ	
		x_0	x_1	x_0	x_1	x_0	x_1
A_1	S_1	S_0	S_7	z_0	z_0	B_1	C
A_2	S_2	S_2	S_0	z_0	z_0	A_2	B_1
	S_3	S_3	S_0	z_0	z_0	A_2	B_1
	S_5	S_5	S_0	z_0	z_0	A_2	B_1
B_1	S_0	S_1	S_7	z_1	z_1	A_1	C
B_2	S_4	S_1	S_4	z_1	z_1	A_1	B_2
	S_6	S_1	S_6	z_1	z_1	A_1	B_2
C	S_7	S_0	S_1	z_1	z_0	B_1	A_1

OSAKA UNIVERSITY

６回目の授業終了

授業終了

皆さん
今日はレポート課題はありません