コンピュータ・マテリアル・サイエンス　第２回
―　コンピュータのしくみ（つづき）　―

名古屋工業大学セラミックス基盤工学研究センター　井田　隆

2.6.3　NAND 回路

２つの入力 A, B に対して $\overline{A \wedge B}$ = not (A and B) を出力する回路のことを NAND 回路といいます。NAND 回路は図 2.15 のように，2.6.2 節の AND 回路と 2.6.1 節の NOT 回路を組み合わせれば実現できるわけです。

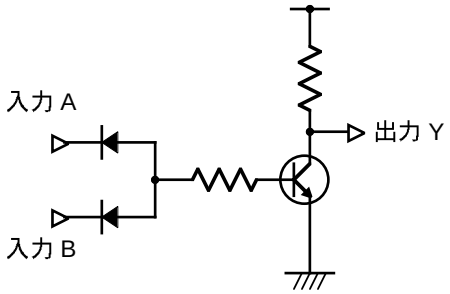

図 2.15　NAND 回路

この回路のはたらきは図 2.16 のように表されます。この回路からは 2.6.2 節の AND 回路のようにあいまいな出力ではなく，はっきりした出力を得ることができるので，確かに $\overline{A \wedge B}$ という論理演算に対応したはたらきをします。このようなはたらきをする論理回路のことを論理ゲートといいます。2.6.1 節の NOT 回路は NOT ゲート（インバータ），この節の NAND 回路は NAND ゲートとも呼ばれます。図 2.17 には NAND ゲートの回路記号を示します。

2.6.4　ゲート素子

論理演算に対応した電子回路をパッケージにしたものをゲート素子と言います。ゲート素子を組み合わせることによって，複雑な論理演算や算術演算も処理することができますし，ある種の記憶素子を実現することができます。代表的な論理ゲートの回路記号を図 2.18 に示します。

図 2.18 のゲート素子は，いずれも NOT ゲートと NAND ゲートの組み合わせで実現できます。図 2.19 にその例を示します。NAND ゲートは NOT ゲートの代わりに使うことができるので，結局 NAND ゲー

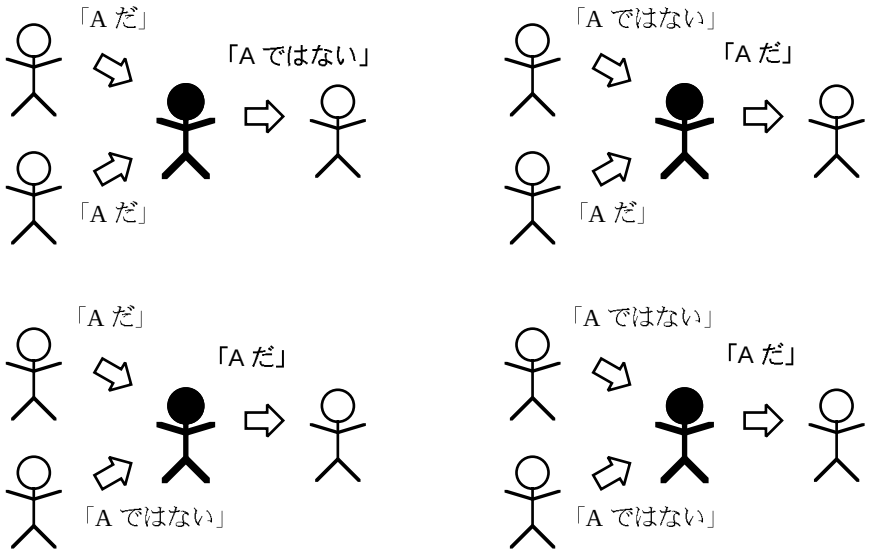

図 2.16 NAND 回路のはたらき

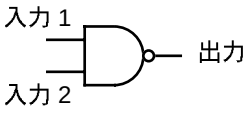

図 2.17 NAND ゲートの回路記号

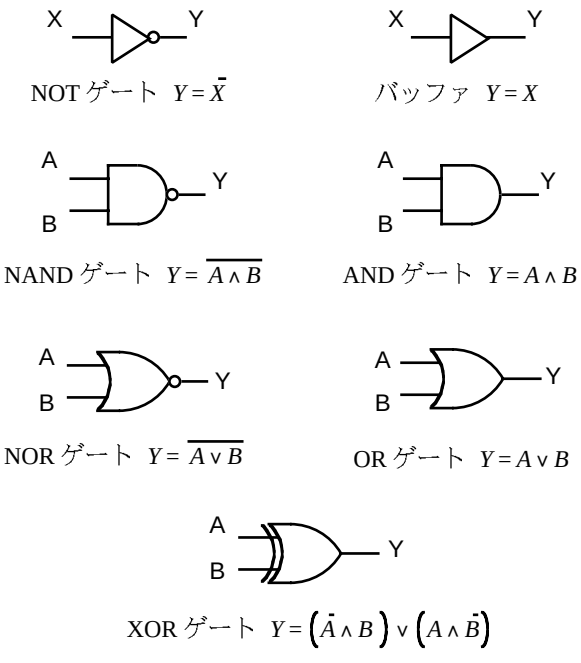

図 2.18 ゲート素子の回路記号

トの組み合わせだけですべての論理演算を実現できることになります。この意味で，2.6.3 節の NAND ゲートは，他のゲート素子の基本となる重要な素子であると言えます。

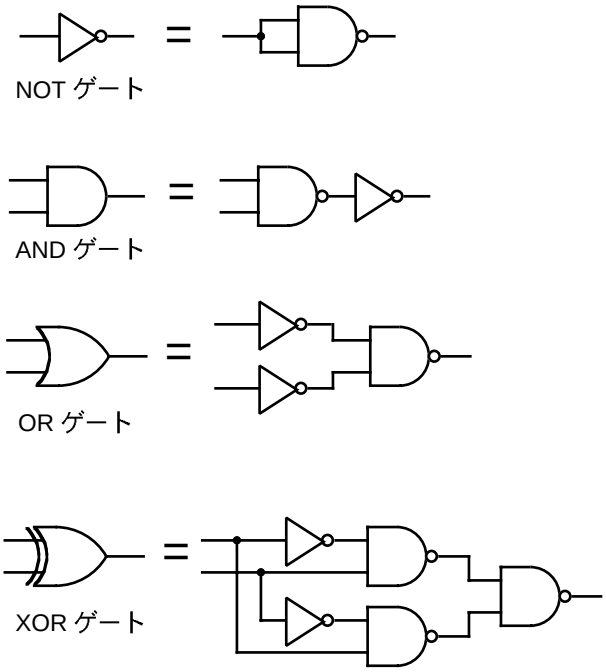

図 2.19 ゲート素子を NAND ゲートと NOT ゲートの組み合わせで作る例

2.7 メモリ

コンピュータはどうやってものを覚えているのでしょうか？電源を入れていない間は磁気ディスクや光ディスクにデータが記録されていることは想像がつくでしょう。これらの装置は「補助記憶装置」と呼ばれます。昔は紙テープやパンチカード，磁気テープなどが補助記憶装置として使われていました。

しかし，データをディスクから読み込んだとしても，主な処理をして次に書き込むまでの間はデータは電子回路に記憶されています。データを記憶しておくための電子回路をメモリと呼びます。原理的には電子回路によるメモリ素子がなくても動作するコンピュータも可能ですが，一般的には補助記憶装置を使ったデータの読み書きよりもメモリ素子を用いたデータの読み書きの方がずっと高速であり，実際上メモリ素子はコンピュータにとって必要不可欠な部品です。

ふつうに使われるメモリ素子には 2 種類あります。

(1) スタティック・メモリ；S-RAM = Static Random Access Memory

(2) ダイナミック・メモリ；D-RAM = Dynamic Random Acces Memory

これら 2 種類のメモリ素子は全然違うしくみになっています。詳しいしくみは後述するとして，これらのメモリ素子の特徴を比較すると表 2.1 のようになります。

表 2.1 S-RAM と D-RAM の比較

素子の種類	読み書きの速さ	構造の複雑さ	集積密度	単価
スタティック	高速	やや複雑	やや困難	やや高価
ダイナミック	やや低速	単純	容易	安価

2.7.1 スタティック・メモリ

スタティック・メモリは論理ゲートの組み合わせで実現されます。典型的な例は以下のように２つの NAND ゲートを組み合わせた回路です。

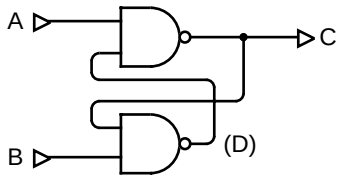

図 2.20 スタティック・メモリ回路

このような回路の出力 C は，２つの入力 A, B によってどのように変化するでしょうか？NAND ゲートのはたらきから，以下の２つの式が成り立ちます。

$$\left\{ \begin{array}{lcl} C & = & \overline{A \wedge D} \\ D & = & \overline{B \wedge C} \end{array} \right. \tag{1}$$

この２番目の式を１番目の式に代入すると，

$$\begin{array}{rcl} C & = & \overline{A \wedge (\overline{B \wedge C})} \\ & = & \overline{A} \vee (B \wedge C) \end{array}$$

という関係が得られます。２番目の式の変形にはド・モルガンの法則を使いました。さらに，A と B の値に応じて，C の値がどのような値を取るかを調べると，以下のようになります。

$$\begin{array}{l} A = 0, B = 0 \text{ のとき，} C = 1 \vee (0 \wedge C) = 1, \\ A = 1, B = 0 \text{ のとき，} C = 0 \vee (0 \wedge C) = 0, \\ A = 0, B = 1 \text{ のとき，} C = 1 \vee (1 \wedge C) = 1, \\ A = 1, B = 1 \text{ のとき，} C = 0 \vee (1 \wedge C) = C \end{array}$$

つまり，

$$C = \left\{ \begin{array}{ll} 1 & (A = B = 0) \\ 0 & (A = 1, B = 0) \\ 1 & (A = 0, B = 1) \\ \text{不定} & (A = 1, B = 1) \end{array} \right. \tag{2}$$

となります。

この回路の C の値をメモリ（記憶）だとみなすことができます。A と B はメモリを書き換えるための信号線です。どういうことかということについて，以下に説明します。

通常の状態では $A = 1, B = 1$ とします。はじめは C の値は不定です。初期化のために「0」というデータを「書き込む」ためには，$A = 1$ に保ったまま $B = 0$ とします。すると，図 2.21 のように C の値が変化します。はじめに，２つある NAND ゲートの下の方を注目してください。この NAND ゲートの入力の一方が 0 になるので，出力 D は必ず 1 になります。すると，上の方の NAND ゲートは２つの入力が両方とも 1 になるので，出力 C が 0 になります。

さて，この $A = 1, B = 0$ の状態から， $A = 1, B = 1$ の状態に戻したらどうなるでしょうか？図 2.22 のように，出力は $C = 0$ のままで変化しないことに注目してください。下の方の NAND ゲートの入力の

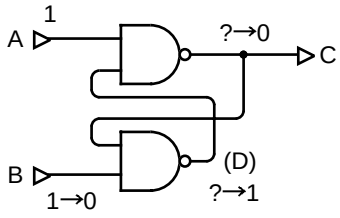

図 2.21 スタティック・メモリ回路に 0 を書き込む。B を 0 にすれば，必ず $C = 0$ となる。

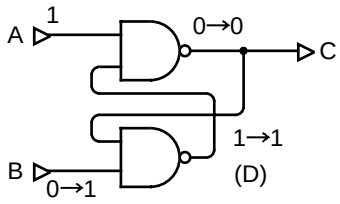

図 2.22 スタティック・メモリ回路に 0 を書き込んだあと，$B = 0$ から $B = 1$ に戻しても，$C = 0$ のままである。(書き込まれたことを憶えている！)

うちの 1 つは，今 $C = 0$ になっているので，出力は $D = 1$ のままです。上の方の NAND ゲートの 2 つの入力は両方とも 1 なので，その出力は $C = 0$ でつじつまがあっていることになります。

次に，このメモリに「1」という値を「書き込む」には，$B = 1$ に保ったまま $A = 0$ とします。すると，図 2.23 のようになります。上の方の NAND ゲートの入力の一方が $A = 0$ となると，その出力は $C = 1$ にならなければなりません。下の方の入力は $B = 1, C = 1$ となるので，その出力は $D = 0$ となります。こ

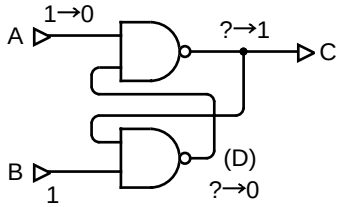

図 2.23 スタティック・メモリ回路に 1 を書き込む。A を 0 にすれば，必ず C が 1 になる。

の後 $A = 0$ から $A = 1$ に戻しても，図 2.24 のように，$D = 0$ なので上の NAND ゲートの出力は $C = 1$ のままです。

このような回路のことをフリップ・フロップ (flip-flop) 回路とかラッチ (ratch) 回路と呼ぶことがあります。フリップ・フロップとは「ひょいとひっくり返す」といった意味合いの言葉です。ラッチはラチェット (ratchet) と同じ意味で，逆回りをしないような機構のついた歯車のことです（自転車でペダルを逆に回す

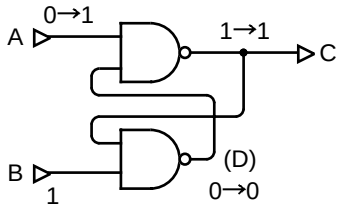

図 2.24 スタティック・メモリ回路に 1 を書き込んだあと，$A = 0$ から $A = 1$ に戻しても $C = 1$ のままになっている。(書き込まれた値を憶えている！)

と空回りするのはラチェット歯車のはたらきによるものです)。この電子回路は，確かに機械的なラチェット機構と同じようなはたらきをしていると言えます。

2.7.2 ダイナミック・メモリ

ダイナミック・メモリ素子は，大雑把に言えばキャパシタ（コンデンサ）に静電気をためることでデータを記憶します。キャパシタの構造は，図 2.25 のように 2 枚の金属板を向い合せに近づけて配置したものです。

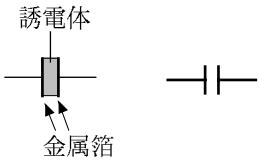

図 2.25 キャパシタ (コンデンサ) の構造 (左) と回路記号 (右)

このキャパシタとダイオード，抵抗を図 2.26 のように組み合わせれば，これがダイナミック・メモリ回路として働きます。キャパシタに電荷がたまっているかどうかで 1 か 0 の値が表されます。ダイナミック・

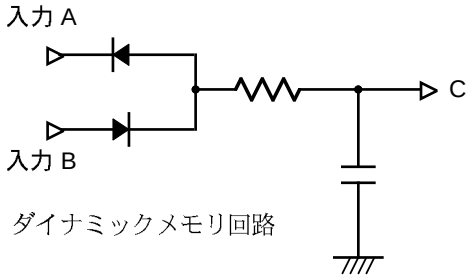

図 2.26 ダイナミック・メモリ回路。通常は $A = 1$ (例えば $+5$ V), $B = 0$ (0 V) とする。

メモリ回路の入力信号は，通常は例えば $A = +5$ V, $B = 0$ V という状態にしておきます。このまま長い間ほうっておけば， $C = +2.5$ V になるはずです。

ダイナミック・メモリに「0」という値を書き込むには，$B = 0$ を保ちながら $A = 0$ とすれば良いだけ

です。キャパシタから A へ電流が流れることにより放電して，$C = 0$ という状態になります (図 2.27)。この後に $A = 1$ に戻しても，A から C の方向へは電流が流れにくいので，$C = +2.5$ V という状態にま

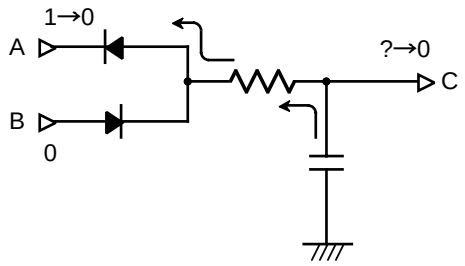

図 2.27 ダイナミック・メモリ回路に 0 を書き込む。A を 1 から 0 に変化させれば，矢印の向きに電流が流れて $C = 0$ となる。

で戻るには時間がかかります。つまり，しばらくは「0」という値を憶えていることになります (図 2.28)。

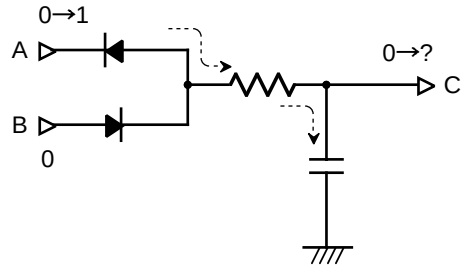

図 2.28 ダイナミック・メモリ回路は 0 という値を憶えている。A を 0 から 1 に戻しても，矢印の向きには電流が流れにくいので，しばらくは $C = 0$ のままである。

次に，ダイナミック・メモリ回路に「1」という値を書き込むには，$A = 1$ を保ちながら $B = 1$ とすればよいだけです。B から電流が流れ込んでキャパシタに電荷が蓄えられ $C = 1$ という状態になります (図 2.29)。この後に $B = 0$ という状態に戻しても C から B の方向へは電流が流れにくいので，やはり $C = 1$ という状態がしばらくは保持されます (図 2.30)。

ただし，通常の $A = 1, B = 0$ の状態でも，キャパシタは少しずつ放電してしまいます。そこで，データを書き込んで一定の時間が経過したら，一度 C の値を読み込んで，もう一度書き込み直すという動作 (リフレッシュ refresh 動作) が必要になります。ふつうのメモリ素子では 1 秒間に数万回くらい (数十 μs に一回) リフレッシュ動作をしているそうです。

2.8 コンピュータによる算術演算

(ディジタル) コンピュータでは数値を 2 進数で表しています。ふつうに市販されているコンピュータは，すべてディジタル・コンピュータですが，アナログ・コンピュータというものもあります。

ここで余談ですが，アナログ回路で計算する一つの典型的な方法を紹介します。例えば 「3 で割る」 という計算をしたいときには，図 2.31 のように，抵抗の大きさが 2:1 になるような 2 つの抵抗を直列につなげば良いだけです。割る数を変えたければ，抵抗の大きさを変えられるような，可変抵抗 (ボリューム) を

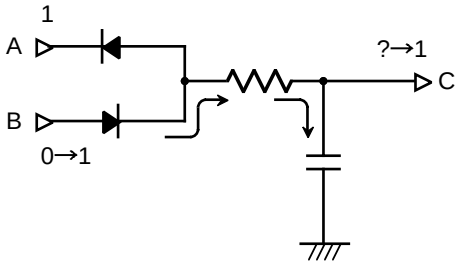

図 2.29 ダイナミック・メモリ回路に 1 を書き込む。B を 0 から 1 に変化させれば，矢印の向きに電流が流れて $C = 1$ となる。

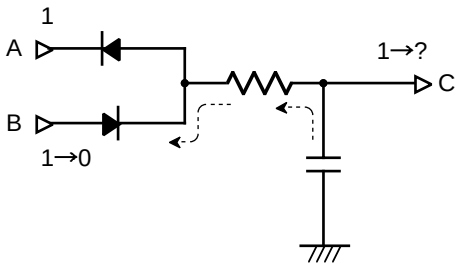

図 2.30 ダイナミック・メモリ回路は 1 という値を憶えている。B を 1 から 0 に戻しても，矢印の向きには電流が流れにくいので，しばらくは $C = 1$ のままである。

使えば良いというわけです。もっと一般的に，アナログ回路で計算をしやすくするための回路をパッケー

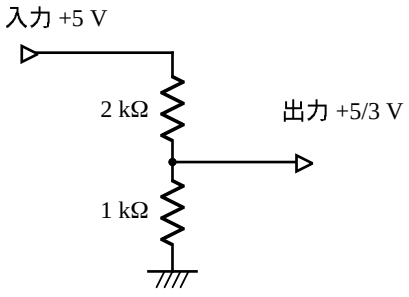

図 2.31 素朴な割り算回路。入力電圧を 3 で割った電圧が出力される。

ジにした素子として「演算増幅器」(通称 オペ・アンプ; operational amplifier) というものがあります。精度にもよりますが，普通のものだと 1 個 100 円くらいで売っています。オペ・アンプを使って加減乗除の計算をするための回路は良く知られています。計測回路などでは「精度は必要ないが，とにかく高速に計算したい」という場合が結構ありますので，知っておいても損はないでしょう。

以下では，ディジタル・コンピュータを使った計算のしかたを紹介します。普通の加減乗除の計算のことを「算術演算 arithmetic operation」と言います。ディジタル・コンピュータでは数値を２進数を使って表します。２進数の算術演算は，前に説明した「論理演算 logical operation」の組み合わせで実現できます。

2.8.1 コンピュータによる足し算

２進数の１桁は 0 か 1 の値しかとりません。この２進数での１桁のことを 「1 ビット bit」といいます。はじめに,「1 ビットの足し算回路」について考えます。つまり,

$$\begin{aligned} 0+0 &= 0 \\ 0+1 &= 1 \\ 1+0 &= 1 \\ 1+1 &= 2 \end{aligned}$$

という４通りの場合について正しい答えを出力するための回路です。10 進数での 2 という数は２進数だと 10 と表されます。ですから，上の例を２進数で表現すると，

$$\begin{aligned} 0+0 &= 00 \\ 0+1 &= 01 \\ 1+0 &= 01 \\ 1+1 &= 10 \end{aligned}$$

と書けます。

このような計算をどのような電子回路で実現できるでしょうか？おおまかには，２本の入力信号線と２本の出力信号線を備えた回路になるはずです。つまり，足し算の２つの入力を A, B という２本の入力信号線に，出力のうち「$2^0 = 1$」の桁の数字を Y_0 という出力信号線，「$2^1 = 2$」の桁の数字を Y_1 という出力信号線に割り当てます。全体としては，図 2.32 のような形になるわけです。

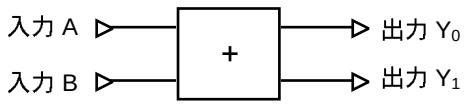

図 2.32 1 桁の足し算回路。Y_0 は 2^0 の桁の数，Y_1 は 2^1 の桁の数を出力する。

つぎに，この回路の中身を考えましょう。出力 Y_0 と Y_1 とに分けて考えます。

出力 Y_0 と入力 A, B の間には表 2.2 のような関係があります。この関係を論理式を使って書くために，次のように考えます。表 2.2 から,「出力が 1 になる」のは,「$A = 0$ かつ $B = 1$」の場合と,「$A = 1$ かつ $B = 0$」の場合との２通りあります。これを式で表せば，

$$Y_0 = (\bar{A} \wedge B) \vee (A \wedge \bar{B}) \tag{3}$$

となります。この関係は，図 2.18 の論理ゲートのところに出てきた「排他的論理和 XOR」と同じですね。つまり，XOR ゲートを使えば一桁の足し算の $2^0 = 1$ の桁を得る回路になるわけです。

つぎに，出力 Y_1 と入力 A, B の間には表 2.3 のような関係にあります。これを式で表せば，

$$Y_1 = A \wedge B \tag{4}$$

表 2.2 1ビットの足し算回路, $2^0 = 1$ の桁の出力 Y_0

A	B	Y_0
0	0	0
0	1	1
1	0	1
1	1	0

表 2.3 1ビットの足し算回路, $2^1 = 2$ の桁の出力 Y_1

A	B	Y_1
0	0	0
0	1	0
1	0	0
1	1	1

となります。この関係は,「論理積 AND」そのものですね。つまり，AND ゲートを使えば 1 桁の足し算の繰り上がりの部分，$2^1 = 2$ の桁を得る回路になるわけです。

これらのことをまとめれば，1ビットの足し算をするための回路としては全体としては，図 2.33 のような構成になるわけです。

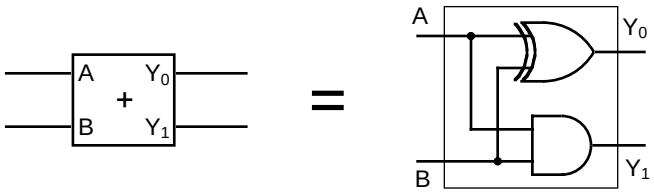

図 2.33 1ビットの足し算回路 (2)。XOR ゲートと AND ゲートの組み合わせで実現できる。

それでは次に，任意の桁数の足し算回路について考えましょう。足し合わせる 2 つの数が両方とも 2 進数で N 桁だとします。つまり，2つの数 A, B がそれぞれ，

$$A_{N-1}A_{N-2}\cdots A_j\cdots A_1A_0, \tag{5}$$

$$B_{N-1}B_{N-2}\cdots B_j\cdots B_1B_0, \tag{6}$$

のように表されるとします。これらを足し合わせた数 Y は，最大でも $N+1$ 桁にしかなりません。これを

$$Y_NY_{N-1}\cdots Y_j\cdots Y_1Y_0, \tag{7}$$

と書くことにします。足し算の結果の 2^j の桁の数字 Y_j は，A_j と B_j，それと 2^{j-1} の桁の繰り上がりがあるかどうかによって決まります。2^{j-1} の桁の繰り上がりがあるかどうかを C_{j-1} と表すことにすれば，出力 Y_j について以下の表が得られます。前に求めた「1ビットの足し算回路」は入力が 2 つだったのですが，ここで求めたいのは,「3つの入力を持った1ビットの足し算回路」ということになります。「2 入力 1 ビット足し算回路」を図 2.33 の記号で表すことにすれば，これを使って「3 入力 1 ビット足し算回路」を図 2.34 のように組み立てることができます。この「3 入力 1 ビット足し算回路を」を必要な数だけ集めれば，任意のビット数の算術和を求める回路が作られます（図 2.35)。

表 2.4 複数桁の足し算回路, 2^j の桁の出力 Y_j と繰り上がり C_j

C_{j-1}	A_j	B_j	Y_j	C_j
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

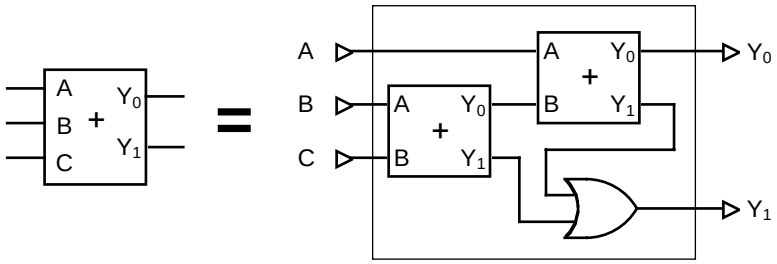

図 2.34 3入力1ビット足し算回路

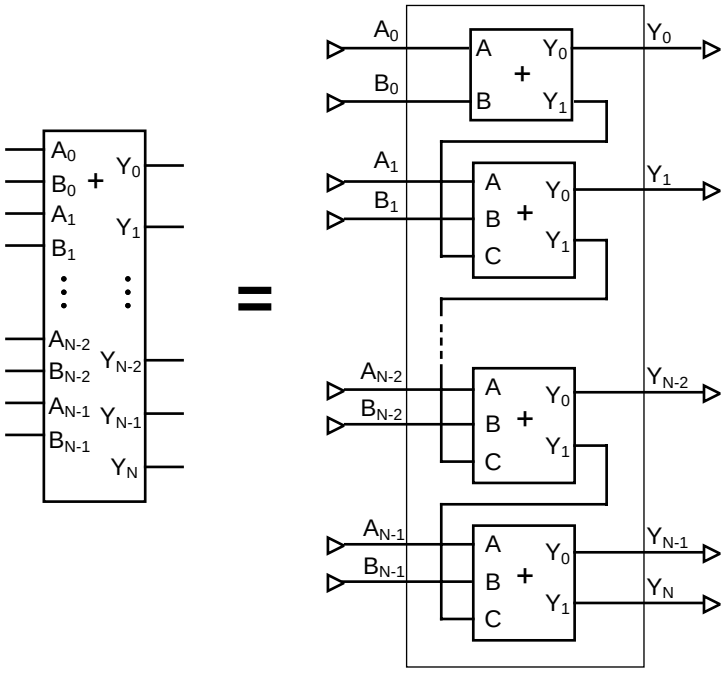

図 2.35 2入力 N ビット足し算回路