このＰＤＦは，ＣＱ出版社発売の「改訂新版 定本 ディジタル･システムの設計」の一部分の見本です．
内容･購入方法などにつきましては以下のホームページをご覧下さい．
<http://shop.cqpub.co.jp/hanbai/books/38/38921.htm>

第 1 章

Digital回路の基礎

物理現象を実際に扱うときは，実際上はすべてdigital（ディジタル）値で行っています．すなわちanalog（アナログ）値といえども，その値を読み取って人に伝えるには，一定の有効数字で打ち切った数字の羅列にするしかないのです．

見本

1.1　Digitalとは

Digital（ディジタル）という言葉は著者が大学を卒業した半世紀ほど前には，専門用語だった記憶があります．もちろんそれに対比して使われているanalog（アナログ）という言葉も専門用語でした．

ここではまず，analogとdigitalの違いを温故知新してみます．もちろんすでにご存じの方は先に進んでください．

1.1.1　Analogとdigital

例題1.1　右脳思考

人間の脳の機能を説明する通説として**図**1.1のように，右脳を働かせることができる人は思考がdigital的ではないという議論があるが，それについてどう思うか．

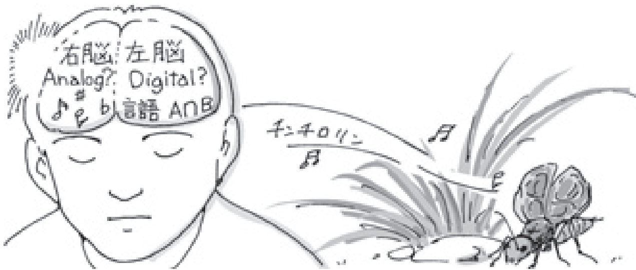


図1.1　**Analog脳とdigital脳？**

［**解**］

この通説では「右脳はanalog的，情緒的，pattern（パターン）的思考を行い，左脳はdigital的，論理的，言語的思考を行う」ということですが，右利きの人の数%と左利きの人の約半数には右脳に言語に関する機能があるそうです．また，日本語を母国語とする人は擬態語や擬声語が発達しているので，虫の音などの自然界の音を言語として認識しているそうです．

Logic（論理）ということばは言語や真理を表す古代ギリシャ語の $\lambda o \gamma o \zeta$（ロゴス）から来ていると言われています．古代から言語は論理的であると考えられていたのでしょう．

実際にanalogとかdigitalということば自体は，電子工学（Electronics（エレクトロニクス））や情報科学（IS：Information（インフォメーション） Science（サイエンス））・情報技術（IT：Information Technology（テクノロジ））の世界に限らず，広く一

般に使用されています．さらにこれを拡張して，人間にはanalog人間とdigital人間がいるというように，人間の性格を論じるのに用いることもあるようです．

＊　＊　＊

本章ではこのようなあいまいな意味ではなく，電子工学や情報科学・情報技術で一般に使われるanalog，digitalの意味について考えてみることにします．Analogとdigitalは，ある注目する対象がもつ数量をどう表現するか，またその扱いをどうするかの方法を論じたものといえます．

例題1.2　Analogとdigitalの違い

Analogとdigitalはどう違うのか，例を出して考えよ．

［**解**］

Analogは**表1.1**に示すように，対象のもつ数量を長さ，面積，角度，電圧，電流などの連続的に変化する物理量によって表す方式をいいます．そして，この連続した物理量をanalog量と呼んでいます．

このようにanalogは，対象のもつ数量を物理量に相似させて表します．Analogという単語はもともと類似物（相似物）という意味の名詞です．一方digitalはdigit（ディジット）（指）から派生した形容詞で“数字で計算をする”という意味です．

表1.1　Analog表示とdigital表示

数量の表現方法	Analog表示（物理量による）	Digital 表示（数値による）
身　長	測定した物差しや柱のキズの位置	176.3 cm（文字表記）

図1.2　Digital表示では単位量が必要

Digitalでは対象のもつ数量を数値によって表現します．文字による数量の表記や計算は基本的にdigital処理といえます．すなわち数量を一定桁数の数字の羅列にして表記し，計算をしているのです．

たとえば，表現する対象が学生の数や自動車の台数のようにその量が離散的で一つずつ数え上げられれば，その値をそのまま数字で表します．しかし，時間や水量のように連続的な量ならそのままでは表せないので，**図**1.2のように，分，秒やm^3のようなある単位量を決めてその何倍という形で表します．

このような，とびとびの有限の桁数からなる数値をdigital量と呼びます．ただ，一般的にはもう少し意味を広げて，単に連続的な量をanalog量，離散的な量をdigital量ということもあります．

＊　＊　＊

1960年代まではanalog computer（コンピュータ）というものが世の中に存在しました．それは計算すべき数量を電圧などの物理量に対応させ，電子回路によって電圧間の加算などを行って計算処理する方法でしたが，今では博物館の陳列物になっています．

例題1.3　Analog時計とdigital時計

Analogとdigitalの違いがよく例になる時計の表示について考えよ．

［**解**］

時計自体は日時計や水刻(水時計)でもないかぎり，チック・タックとゆっくりした音を出して動く振り子時計はdigital的に動いています．ただ表示方式が違います．

時計の表示方式は，従来からの針式すなわちanalog式と，多数の素子を集積したIC(Integrated Circuit（インテグレーテッド・サーキット）：集積回路)化とともに普及したdigital式に大別できます．

(a) Analog式

(b) Digital式

図1.3
時計のanalog表示とdigital表示

(1) Analog表示式の時計は，**図**1.3(a)のように，時刻を長針と短針がそれぞれ作る角度(Analog量)で対応させて表す.

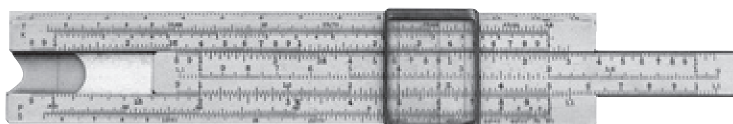
(2) Digital式の時計では**図**1.3(b)のように，時刻を秒ないし分を単位として離散的に数字(Digital量)で示す.

例題1.4　計算尺とそろばん

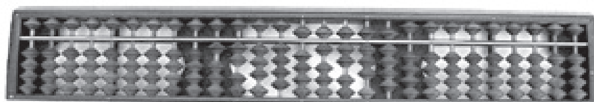
今はあまり見ることもない計算用の器具である計算尺とそろばんについて，analogとdigitalの演算処理方式の違いを考えよ.

［**解**］

図1.4に示す計算尺とそろばんが例として考えられます．計算尺は最近はほとんど使われませんが，乗除算や平方根，べき乗の計算などが簡単にできます．1970年代に電卓が普及するまでは，理工系の研究者・技術者や学生にとっては必携の道具でした.



(a) 計算尺(2×3=6 の計算をしているところ)



(b) そろばん(算盤：中国から伝わった．1～9 の数値が入れてある)

図1.4　**計算尺とそろばん**

計算尺の特徴

(1) 計算しようとする数値を対数目盛上の長さ(Analog量)に対応させ，その長さを継ぎ足したり差し引いたりする操作によって計算結果を求める.

(2) 目盛りを合わせるだけで，乗除算や平方根・べき乗の計算の答を誰でも簡単に

そろばんの特徴

(1) 数値をそのまま玉の位置(Digital量)で表し，玉の上下操作で計算する.

(2) 熟練者は別として，複雑な計算は，計算に時間がかかる．平方根やべき乗の計算はむずかしく，これをやろうとすると非常に複雑な操作を要す.

得ることができる.

(3) 刻まれている目盛以上の精度は目分量で読み取るしかない.

(4) ちょっとでもゆがんでいたり，Cursor(カーソル)ががたついていると精度が悪くなり，場合によっては使用に耐えられなくなってしまう.

(5) 計算の途中でズレが生じると計算結果も狂ってくる.

(3) 計算の精度を必要とするなら，機構的に桁数の許す限り何桁でも得られる.

(4) そろばんは形さえ整っていれば一応使用に耐える．多少玉の滑りが悪くても計算の精度は同じように得られる.

(5) 計算中操作が乱れて玉が正規の位置より多少ずれても，玉の上下の位置がどちらか判別できたなら値は特定できるので，計算結果に影響を与えない.

以上のような計算尺とそろばんの特徴は，電圧値などをそのままdataとして使うanalog処理と，電圧の高低を基準とするdigital処理の特徴にもなっています.

例題1.5 Analog処理とdigital処理の比較

Analog処理と対比してdigital処理の特徴を整理してみよ.

［**解**］

表1.2にanalog処理とdigital処理の特徴をまとめたものを示します.

表1.2 **Analog処理とdigital処理の特徴**

	Analog 処理	Digital 処理
利点	• 処理過程が直感的	• 外乱や自己の不安定性の影響に強い，digital量を2値的に表現した場合は顕著 • 誰がやっても同じ答が出る • 桁数を多くとれば精度をそれだけ高められる
欠点	• 精度が出ない • Analog IC などの素子の性能が悪いと直接的に装置全体の性能に影響 • Program(プログラム) 処理が難しい	• 少しでもこみいった処理を行おうとすると操作が複雑となり，それを扱う装置や機械も複雑化してしまう • 一つ一つの数値を数学的・論理的に処理するので，原理的には処理が長くかかる • 複雑な処理はprogramを作成しなければならない

不安定性に強いということは，digital処理装置においてその構成する素子の性能にそれほど依存しないということです．すなわち，装置を構成する素子であるdigital ICの性能が多少劣化していても，論理どおりに機能さえすれば装置は正しく動作します.

＊　＊　＊

第 2 章

Bool代数とDigital回路の表記

— 論理式・回路図・HDL —

歴史的にはdigital回路はBool(ブール)代数があってはじめて実用的に使えるようになりました．もしBool代数がなければdigital回路を構築するのはたいへんでした．

このBool代数は論理学と直接的な関係があります．Bool代数は集合論のごく基本的な知識があれば，だれでも直感的に理解できます．では，しばらく頭の体操にお付き合いください．

2.1　Digital回路の機能へのBool代数の導入

2.1.1　Digital回路の機能を数式と真理値表で表現

例題2.1　回路機能の数式表現

Analog回路およびdigital回路は，その動作をどのように解析したり設計したりするのか．

［**解**］

Analog回路では回路の機能を行列で表します．Transistor回路の場合は前章で少し触れたように，行列の要素を**図**2.1(a)のようにh-parameterで与えて機能を規定します．しかし，行列という言葉を聞くだけで挫折感を覚える人もいるかも知れません．

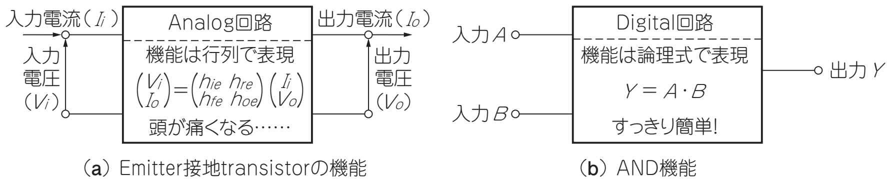


図2.1　**回路の機能を式で表現**

一方digital回路では，前章で示したようにdigital信号をHとLという二つの状態で表現し，このHとLを扱うdigital素子を組み合わせて種々の機能を実現します．Digital回路では，この回路の機能を表現するために，二つの状態HとLに二つの数字‘1’と‘0’を対応させると，**図**2.1(b)のように数学的な取り扱いができます．Analog回路よりは見た目には簡単な式で記述されていることが分かります．

このような2値を扱う学問にはBool代数(Boolean algebra（ブーリアン・アルジェブラ）)というものがあり，digital回路の種々の機能の記述に，このBool代数が導入できます．一般にdigital回路の機能を論理機能(Logical function（ロジカル・ファンクション）)といってます．すなわちdigital回路の機能はBool代数の関数(論理式)で表されるということです．論理機能はBool代数の計算式になりますから論理演算ともいいます．

例題2.2　真理値表による表現

AND，OR，NOTという論理演算それぞれについて，入力の組み合わせによる出力状態の表を作成して記述せよ．

［**解**］

Bool代数では2値を‘1’と‘0’で表し，以下の三つの演算を定義します．

名　称	本書での記号	英語名
論理積	AND，“・”	Logical product（プロダクト）
論理和	OR，“＋”	Logical sum（サム）
否定	NOT，“ー”	Negation（ネゲーション）

表2.1
Bool代数の演算規則

AND	OR	NOT
1・1＝1	1＋1＝1	$\overline{1}=0$
1・0＝0	1＋0＝1	$\overline{0}=1$
0・1＝0	0＋1＝1	$\overline{\overline{1}}=1$
0・0＝0	0＋0＝0	$\overline{\overline{0}}=0$

表2.1はこのAND，OR，NOTについて，入力の‘1’，‘0’すべての場合のBool代数の演算規則を表したものです．

この演算規則は10進数の九九などのように値そのものに対する規則ですが，通常の代数と同じく各論理機能の入力や出力に‘1’，‘0’のどちらかの値をとる変数A，B，Yを導入することができます．これを論理変数(Logical variable（バリアブル）)といいます．

すると二つの2値変数A，Bに対しては**表2.2**のように演算結果Yを表にまとめることができます．このように論理演算の各入力変数のすべての入力の組み合わせに対する出力を表にまとめたものを真理値表(Truth table（ツルース・テーブル）)といいます，

表2.2
基本論理演算の真理値表

入力		出力
A	B	$A\cdot B$
0	0	0
0	1	0
1	1	1
1	0	0

(a) AND(論理積)

入力		出力
A	B	$A+B$
0	0	0
0	1	1
1	1	1
1	0	1

(b) OR(論理和)

入力	出力
A	$\overline{A}$
0	1
1	0

(c) NOT(否定)

表2.2(a)では論理積についての出力値を示していますが，'1'を真(True)，'0'を偽(False)とみなせば，AとBが真のときに結果($A \cdot B$)が真となるので，論理的にANDの操作となっています．**表2.2**(b)では，AかBの少なくともどちらか一方が真ならば，結果は真となるので，ORの操作を行っていることが分かります．

表2.2(c)は，Aが真なら偽，偽なら真となり，明らかにNOTの操作となっています．このようにBool代数の基本論理演算は，それぞれ論理操作に対応していることが分かります．

例題2.3　論理演算の式

Bool代数の演算で書かれた式を論理式(Logical formula)という．二つの論理式を等号で結んだ恒等式は，どのようなものが成り立つか調べよ．

［**解**］

恒等式とは変数の値にかかわらず等号(＝)の左右にある式の値が常に等しくなる式をいいます．Bool代数は代数の一種ですから，**表2.3**のように代数学上の恒等式はそのまま成立します．

表2.3　Bool代数の恒等式

交換則	$A \cdot B = B \cdot A$ $A + B = B + A$
吸収則	$A \cdot (A + B) = A$ $A + A \cdot B = A$
結合則	$(A \cdot B) \cdot C = A \cdot (B \cdot C)$ $(A + B) + C = A + (B + C)$

分配則	$A \cdot (B + C) = A \cdot B + A \cdot C$ $A + B \cdot C = (A + B) \cdot (A + C)$
deMorganの法則	$\overline{A \cdot B} = \overline{A} + \overline{B}$ $\overline{A + B} = \overline{A} \cdot \overline{B}$
'1'と'0'に関する法則	$A \cdot 1 = A$，$A \cdot 0 = 0$ $A + 1 = 1$，$A + 0 = A$

吸収則やdeMorgan(ド・モルガン)の法則があったり分配則の一部が実数の代数と異なる点は，視点を変えて後の**例題2.10**で説明する集合論から考えれば容易に理解できます．

これらの式が成り立つことは通常の代数と同じように証明できますが，変数に'1'，'0'を代入してすべての場合について検証を行うことでも，容易に確かめることができます．これは場合の数が有限であるBool代数の大きな特徴といえます．

なお，これらの式で混乱が起こらない限り，論理積の記号"・"は省くことがありますが，本書では省くと2文字以上からなる変数と間違いやすいので，省かずに記述しています．また一般に式の中にこの三種の演算が含まれるとき，"¯"，"・"，"＋"の順に演算が優

先されます．もちろん( )内の演算が最優先となることは，普通の代数とまったく同じです．

例題2.4 deMorganの法則

deMorganの法則が成り立つことを論理式にすべての場合の‘1’，‘0’を代入することで示せ．

［**解**］

実際に**表**2.4のように各変数のすべての‘1’，‘0’の組み合わせについて式の値を計算すると，deMorganの法則の式の両辺の値は常に等しいことが証明できます．

表2.4 真理値表によるdeMorganの法則の証明

A	B	$\overline{A}$	$\overline{B}$	$A \cdot B$	$\overline{A \cdot B}$	$\overline{A}+\overline{B}$
0	0	1	1	0	1	1
0	1	1	0	0	1	1
1	1	0	0	1	0	0
1	0	0	1	0	1	1

(a) $\overline{A \cdot B}=\overline{A}+\overline{B}$の証明

A	B	$\overline{A}$	$\overline{B}$	$A+B$	$\overline{A+B}$	$\overline{A}\cdot\overline{B}$
0	0	1	1	0	1	1
0	1	1	0	1	0	0
1	1	0	0	1	0	0
1	0	0	1	1	0	0

(b) $\overline{A+B}=\overline{A}\cdot\overline{B}$の証明

例題2.5 論理式の簡単化

次の等式が成り立つことを示せ．

(1) $A \cdot A = A$，$A + A = A$(べき等則)

(2) $A \cdot \overline{A}$ = ‘0’ (排他律)，$A + \overline{A}$ = ‘1’ (排中律)

［**解**］

論理式の簡単化の方法は，A = ‘1’ とA = ‘0’ のそれぞれの場合について，**表**2.1の演算規則と照らし合わせて考えると，明らかであることが分かります．

例題2.6 吸収演算の練習

複数の基本論理演算からなる論理式は，Bool代数の演算法則にしたがって，より簡単な形に変形することができる．

吸収演算の法則を用いて，次の式が成り立つことを示せ．

(1) $A + A \cdot B = A$

(2) $A + \overline{A} \cdot B = A + B$

(3) $(A + B) \cdot (A + \overline{B}) = A$

［解］

(1) $A + A \cdot B = A \cdot 1 + A \cdot B = A \cdot (1 + B) = A$

(2) $A + \overline{A} \cdot B = A \cdot (1 + B) + \overline{A} \cdot B = A + A \cdot B + \overline{A} \cdot B = A + (A + \overline{A}) \cdot B = A + B$

(3) $(A + B) \cdot (A + \overline{B}) = A \cdot A + A \cdot \overline{B} + B \cdot A + B \cdot \overline{B}$

$= A + A \cdot \overline{B} + A \cdot B = A + A \cdot (\overline{B} + B) = A + A = A$

＊ ＊ ＊

(1)では$A = A \cdot 1$を用い，分配則により$A + A \cdot B$は$A \cdot 1 + A \cdot B$と置き換えることができます．

この式をかっこでくくると$A \cdot (1 + B)$となります．すると$1 + B = 1$であるため，この式は$A \cdot 1 = A$となり，

$A + A \cdot B = A$

が成り立つことが分かります．

(2)では(1)で証明されたように$A \cdot (1 + B) = A$となるので，まずAを$A \cdot (1 + B)$に置き換えることで，

$A + \overline{A} \cdot B$を$A \cdot (1 + B) + \overline{A} \cdot B$

と変形します．次に$A \cdot (1 + B)$の部分を展開して，

$A \cdot (1 + B) + \overline{A} \cdot B$を$A + A \cdot B + \overline{A} \cdot B$

という形に変形します．$A \cdot B + \overline{A} \cdot B$は分配則$A \cdot B + A \cdot C = A \cdot (B + C)$により，

$B \cdot A + B \cdot \overline{A} = B \cdot (A + \overline{A})$

とBでくくることができ，

$A + A \cdot B + \overline{A} \cdot B = A + (A + \overline{A}) \cdot B$

となります．最後に，$A + \overline{A} = 1$であるため，

$A + (A + \overline{A}) \cdot B = A + 1 \cdot B$

第 3 章

いろいろな組み合わせ論理回路の実現

この章からHDLと回路図での記述を主とします．HDLはVerilog HDLで記述し，VHDLの記述も極力加えます．Digital回路の設計技術のむずかしさは，じつはかなりの部分が組み合わせ論理回路の設計に属しています．複雑な論理機能は基本的には組み合わせ論理回路で決定し，その結果をflipflopやregister, memoryに保存するだけです．

組み合わせ論理回路も複雑な論理の多段接続による実現だけでなく，LSIの大規模化に伴い，並列処理による高速化や，従来実現をあきらめていたprogram処理のhardware化へも手が出せる時代になりました．このようにhardwareとsoftwareのseamless(シームレス)（継ぎ目がない）化が進行していますが，hardwareにはsoftwareが弱い同時動作という特徴があります．

3.1　Dataを選択する回路—Selector

3.1.1　二つのdataや信号から一つを選択する回路

> **例題3.1　信号の選択**（従来の論理式とHDL）
>
> 入力信号AとBを選択信号Sの‘1’，‘0’で切り替えて出力Yとして出したい．Bool代数の論理式とHDLの論理式とで記述せよ．

［**解**］

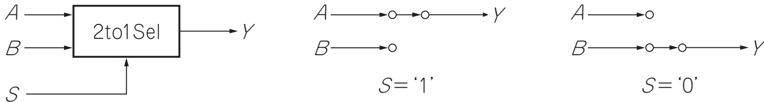


図3.1　1 bit信号のselector

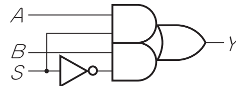


図3.2
信号selectorのMIL記号回路図

図3.1に示す回路は通常，選択信号Sが‘1’か‘0’かによって二つのdataや信号のどちらを選択するかを決定します．

図3.1はMIL記号で描くと**図3.2**のようになります．なお，ANDとORが連続して描かれていますが，これは間の線を省略したものです．

Digital systemでは二つのdataや信号のどちらかを選択して監視したり，利用したりする用途がたくさんあります．この機能はdata selector（セレクタ）とかdata multiplexer（マルチプレクサ）と呼ばれています．

● 従来の論理式による記述

$$Y = A \cdot S + B \cdot \overline{S}$$

● Verilog HDLの論理式による記述

```
module sample3_1(Y, A, B, S);  // 回路の名称と入出力信号の定義
  input A, B, S;             // 信号の宣言，同じものをまとめてもよい
```

```
    output Y;
    assign Y = A & S | B & !S;    // 論理式を代入(式ごとに;が必要)
  endmodule                       // 終了(;は不要)
```

論理式などを別の信号に代入(物理的に接続)するには`assign`文を使います．Verilog HDLでは，回路を定義する`module`文の`()`の後には`;`を付け，設計の最後を示す`endmodule`文の後に`;`は付けません．Verilog HDLではいくつかの回路設計(Module)を一つのfileにまとめることができます．

● **VHDLの論理式による記述**(宣言文の一部は省略)

```
  entity sample3_1 is                // 信号の宣言部
    port(A: in std_logic;            // 一つずつ宣言する
         B: in std_logic;
         S: in std_logic;
         Y: out std_logic);
  end sample3_1;                     // 宣言部終了
  architecture rtl of sample3_1 is
  begin
    Y <= (A and S) or (B and not S); // 論理式を代入(式ごとに;が必要)
  end rtl;                           // 終了(;が必要)
```

VHDLでは代入には`<=`を使い，文は基本的に`;`で区切ります．また入出力portの宣言は信号やdataごとに行います．なお，宣言文の一部は省略してあります．

論理演算の優先順位は`not`だけが高く，ほかの`and`，`or`，`nand`，`nor`，`xor`は同じです．Verilog HDLのようにANDはORに優先しないので，かっこで括る必要があります．

なお，本書の本文ではAND，OR，NOTは大文字を使っていますが，各設計中のVHDLの予約語はVerilog HDLと同様に小文字で書きます．

回路設計が思った通りになっているかどうかは動作simulationを行って検証します．本書ではsimulationはsimulatorの波形editorで検査波形を作成して実行します．

図3.3に示すように，選択信号Sが'1'，'0'と変わるにしたがって，それぞれA，Bが交互に選択されて出力Yに送られているのが分かります．

なおsimualtionを実行するためのsimulatorの使い方についてはそのsimualtorのhelp(ヘルプ)およびいろいろな解説記事やweb siteを参照してください．

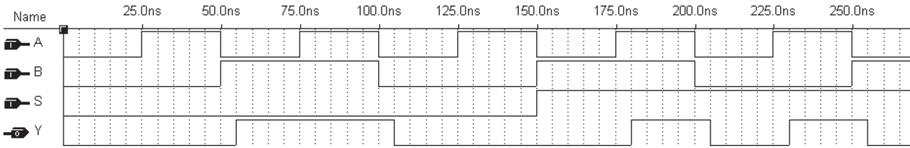


図3.3　信号selectorのsimulation波形

> **例題3.2　Byte dataの選択**（Verilog HDL）
>
> 各8 bitの入力data AとBを選択信号Sで切り替えてdata Yとして出力したい．この回路をVerilog HDLの論理式で設計せよ．

［**解**］

二つ以上の多bitからなるgroup（グループ） dataはbus（バス） dataともいいます．Bus dataをまるごと選択して出力する回路は非常に多く使われています．なお，多bit dataはVHDLではvector（ベクタ）と名付けられています．

従来の論理式ではgroup dataを表現する方法は書籍によってバラバラでした．Verilog HDLやVHDLでは，Cなどのprogramming言語と同じようにその表現形式が決まっています．

Verilog HDLではbus dataは信号番号の上限と下限を`:`で区切って並べ，`[7:0]`と表します．このbusの記述をdataの名称の後に`A[7:0]`と空白を置かずに付けます．Bus dataを宣言するときには，C言語にならって入出力宣言の`input`，`output`が文の先頭に置かれ，data幅の定義，data名称の順に，

```
input␣[7:0]  A;
```

と並べます．宣言文と信号名称との区切り記号は空白␣を使います．

Bus dataの中の一つを単独で指定するときは，bit番号を`A[3]`などと書きます．これを`A3`と略記することはできません．

Verilog HDLでは，bus dataに対する論理演算子はdata同士の対応するbitごとの演算になります．そのため従来の論理式でよく使われるようにbus dataと1 bitの制御信号のANDを一つの論理式で，`A[7:0] & S`と直接に記述することはできません．そこで，bus dataはbitごとに分解して論理演算を施します．

8 bit幅の選択信号`s[7:0]`を宣言して，dataとしてANDを取ることもできますが，複雑になるだけです．

● **Verilog HDLの論理式での記述**

```
module Sample3_2 (Y, A, B, S);          // 回路の名称と入出力信号名
  input  [7:0] A, B;                    // Dataや信号の宣言
  input        S;
  output [7:0] Y;
  assign Y[7] = A[7] & S | B[7] & !S;  // 論理式をassign文で定義
  assign Y[6] = A[6] & S | B[6] & !S;
  assign Y[5] = A[5] & S | B[5] & !S;
  assign Y[4] = A[4] & S | B[4] & !S;
  assign Y[3] = A[3] & S | B[3] & !S;
  assign Y[2] = A[2] & S | B[2] & !S;
  assign Y[1] = A[1] & S | B[1] & !S;
  assign Y[0] = A[0] & S | B[0] & !S;
endmodule                               // 終了
```

例題3.3　Selectorを数式で記述(HDL)

例題3.2のVerilog HDLの記述は同じような式がたくさんあり，たいへん不便である．Verilog HDLやVHDLで論理式を使わないで，数式で記述するにはどうすればよいか.

［**解**］

● **Verilog HDLの条件付き代入文での記述**

```
module Sample3_3 (Y, A, B, S);
  input  [7:0] A, B;
  input        S;
  output [7:0] Y;
  assign Y = (S == 1'b1) ? A : B;   // 条件付き代入文
endmodule
```

Bus dataを選択条件信号Sなどで選ぶときは，Verilog HDLでは条件付き代入文を使う方法があります．条件が成立したときの代入値は“?”の後に，成立しなかったときの代入値は“:”の後に置きます.

`assign` dataや信号=(条件式または論理値)? 条件成立時の値:条件不成立時の値;

この記述法はC言語にある条件演算子と同じですが，ちょっと見ただけでは理解しにく

い書き方なので，C言語に慣れていない人は，Verilog HDLにある程度習熟してから使った方がよいかもしれません．

＊　＊　＊

条件付き代入文では条件式は`()`に入れなくても使えます．しかし式が見にくくなるので，`()`を使うほうがよいでしょう．

選択信号*S*を比べる相手には，この設計では論理値を表す`1'b1`を使っていますが，10進数の値として1を使い，`(S==1)`とも書いてもかまいません．

`()`内の判断式は，変数*S*が1 bit幅のdataのときは，`(S==1)`や`(S==1'b1)`としなくても，*S*をbooleanと見立てて，`(S)`としても同じになります．ただしこれは，C言語の記法をそのまま持ち込んだ形なので，論理値であることが明確な場合以外は使わないほうがよいかもしれません．

● **VHDLの`when`～`else`文での設計**（宣言文の一部は省略）

```
entity Sample3_3 is
  port(A: in std_logic_vector(7 downto 0);
       B: in std_logic_vector(7 downto 0);
       S: in std_logic;
       Y: out std_logic_vector(7 downto 0));
end Sample3_3;
architecture Sample3_3 is
begin
  Y <= A when (S = '1') else B;
end rtl;
```

VHDLでは，`when`～`else`文構造がVerilog HDLの条件付き代入文と同じように使えます．

Dataや信号 `<=` 条件成立時の値 `when(` 条件式 `)else` 条件不成立時の値 `;`

この条件式はかならず`()`に入れて記述します．なお条件が等しいことを調べる等号は`==`ではなく，`=`を一つだけ書きます．

＊　＊　＊

VHDLでは1 bitの論理値は`'1'`あるいは`'0'`と記述します．なお条件文はVerilog HDLのように`(S)`と省略することはできません．

多bitからなるgroup dataはVHDLではvectorとして扱い，下記のように宣言時にその

第 4 章

演算用の回路

2進数での演算，それはdigital回路が使われはじめた当初から要求された機能でした．2進数でも10進数と同じく位取り記法を使います．

インドにおける零の発見とそれによる位取り記法の概念がなかったアジアでも，和算に代表される数学はかなり高度な水準まで発達しました．しかし，それはやはりとてつもなく難解な学問であったと思われます．2進数の演算は位取り記法の便利さを私たちに如実に分からせてくれます．

4.1 2進数の計算

Digital回路では数値，文字，記号をdataとして符号化して扱います．Digital回路において，数値は2進法(Binary［バイナリ］)によって表示された2進数(Binary number)を用います．われわれが通常使用しているのは，10個の記号(数字)よりなる10進法(Decimal［デシマル］)です．数を表す記号が二つしか存在しない2進法で大きな数を表すにはどうすればよいのでしょうか．

2進数については前にも簡単に触れましたが，ここでまとめて考えます．

4.1.1 2進数によるdataの表現

例題4.1 2進表記

2進数値の小さいほうを×，大きいほうを○とした場合，0～7までの10進数はどう表現できるか．

［解］

表4.1 2進数の位取り

10進数	0	1	2	3	4	5	6	7
2進数	×	○	○×	○○	○××	○×○	○○×	○○○

2進数値を表現する方法は二つの異なる記号さえあれば何でもかまいません．小さいほうを×，大きいほうを○ではなくて，それぞれ“零”，“壱”としても同じです．ただ**表4.1**のような位取り記法を用いなくては大きな数値は表現できません．2進法では通常，Bool代数との整合性を考え，×に‘0’，○に‘1’を対応させます．

2進数から10進数への変換，あるいはその逆は，**表4.1**を拡張した対応表を作っておけばよいのですが，数が大きくなると実用的ではありません．しかし，**表4.2**のように0から15くらいまでの対応は覚えておくといろいろと便利です．4桁の2進数では16種のpatternが表現できるので，10～15に英字のA～Fを対応させた16進表示が実用の世界

表4.2 10/16進数と2進数の対応

10(16)進数	0	1	2	3	4	5	6	7
2進数	0000	0001	0010	0011	0100	0101	0110	0111
10(16)進数	8	9	10(A)	11(B)	12(C)	13(D)	14(E)	15(F)
2進数	1000	1001	1010	1011	1100	1101	1110	1111

ではよく使われます．

ここで非常に重要なことは，同じ‘1’，‘0’を用いていてもBool代数と2進数は似て非なるものだということです．Bool代数は論理機能を記述する数学で，2進数は表現が違うだけで，10進数が従っていた代数と同じ代数の管理下にあります．当然，Bool代数では‘1’＋‘1’＝‘1’ですが，2進数では1＋1＝10です．絶対に「Bool代数は1＋1が2でなく1となる代数」などという皮相的な見方をしないでください．

例題4.2　10進−2進変換

10進数を2進数に変換する方法を考えよ．

［**解**］

10進数を2進数に変換する場合，一般的な変換法則を用いることが有効です．**図**4.1は10進数から2進数への変換を示してあります．これには**図**4.1(a)の2のn乗を引いていく方法(減算法)と，**図**4.1(b)の2で割っていく方法(除算法)があります．減算法では前もっ

(a) 減算法　(b) 除算法

図4.1　**10進数から2進数への変換方法**
(366→101101110)

(a) 加算法　(b) 乗算法

図4.2　**2進数から10進数への変換方法**
(11011001→217)

て2のn乗の値を知っておかなければなりませんが，上から2，3桁分計算すれば概算値が分かるという利点があります．一方，除算法では2で割って余りを並べて逆に読むことになり，最後まで計算しないと桁数さえ分かりません．しかし，2のn乗の値を知らなくても変換できるという利点があります．

例題4.3　2進-10進変換

2進数を10進数に変換する方法を考えよ．

［**解**］

2進数を10進数へ変換するには，**図**4.2(p.173)のように逆に2のn乗を加えていく方法(加算法)と，2倍していく方法(乗算法)があります．2のn乗の値を知っていれば，加算法は簡単です．これに対して乗算法は，多少手間がかかりますが2のn乗の値を知らなくても使えます．筆算では，10進数から2進数への変換には除算法が，2進数から10進数への変換には加算法が手慣れた人には便利です．

＊　＊　＊

2進数を用いた10進数のより容易な表現法として，第3章で少し説明した2進化10進法(BCD：Binary Coded Decimal)があります．

このBCDでは，10進数の各桁を**表**4.3のように2進数で表現します．したがって，たとえば73という数値をBCD符号で表すと，**図**4.3のようになります．このように10進数の1桁ずつを，2進数の4桁ずつで区切って表現しているので，変換が非常に簡単です．しかし，2進数の4桁で表現できる数は0から15まであるので，この方法では10から15まで，

表4.3 2進化10進法

10の進数の各桁	BCD
0	0 0 0 0
1	0 0 0 1
2	0 0 1 0
3	0 0 1 1
4	0 1 0 0
5	0 1 0 1
6	0 1 1 0
7	0 1 1 1
8	1 0 0 0
9	1 0 0 1

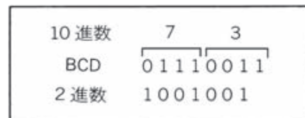


図4.3　10進数73のBCDと2進数による表現

すなわち2進の1010から1111までがBCDでは使用されず，その意味で符号としてむだがあります．

表4.4
3余り符号と3余りGray符号

10進数	3余り				3余りGray			
					A	*B*	*D*	*C*
0	0	0	1	1	0	0	1	0
1	0	1	0	0	0	1	1	0
2	0	1	0	1	0	1	1	1
3	0	1	1	0	0	1	0	1
4	0	1	1	1	0	1	0	0
5	1	0	0	0	1	1	0	0
6	1	0	0	1	1	1	0	1
7	1	0	1	0	1	1	1	1
8	1	0	1	1	1	1	1	0
9	1	1	0	0	1	0	1	0

（a）10進数と3余り，3余りGray

AB \ *CD*	0 0	0 1	1 1	1 0
0 0				0
0 1	4	3	2	1
1 1	5	6	7	8
1 0				9

（b）Gray符号のKarnaugh図

このほかに**表4.4**に示すように，3余り(Excess-3)符号や3余りGray（グレイ）符号があります．3余り符号は数字の0とdataのない場合の0が区別でき，また4との5の間を境にして，‘1’と‘0’を反転させると上下対称となります．3余りGray符号は数字が一つ隣りに移るとき，2進数の四つの桁のうち一つだけ‘1’と‘0’が反転するようになっています．すなわち隣り同士の数は，**表4.4**(b)のKarnaugh図で分かるように，その構成する4 bit中で1 bitのみ異なるBool空間でのHamming距離1の構造となっています．

Digital回路では，数値以外に文字や記号もしばしば扱うことがあります．そのために文字や記号を‘1’，‘0’で符号化しなくてはなりません．これにはいくつか決められた符号があります．一般的にはASCII（アスキー）符号が使われています．また，事務処理用の大型計算機ではEBCDIC符号が使われていたこともあります．日本でもJIS(日本工業規格)で，カタカナやAlphabet（アルファベット）(Roma（ローマ）文字)の符号が決められています．JISに決めた8 bitの符号を**表4.5**に示します．この表から，たとえば英文字Aは“01000001”という符号で与えられることが分かります．

表4.5 8 bit情報交換用符号

上位4 bit

	Control		英数								カナ					
下位＼上位	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
0000	NUL	DLE	SP	0	@	P	'	p			未定義	ー	タ	ミ		
0001	SOH	DC_1	!	1	A	Q	a	q			。	ア	チ	ム		
0010	STX	DC_2	”	2	B	R	b	r			「	イ	ツ	メ		
0011	ETX	DC_3	#	3	C	S	c	s			」	ウ	テ	モ		
0100	EOT	DC_4	$	4	D	T	d	t			、	エ	ト	ヤ		
0101	ENQ	NAK	%	5	E	U	e	u			・	オ	ナ	ユ		
0110	ACK	SYN	&	6	F	V	f	v			ヲ	カ	ニ	ヨ		
0111	BEL	ETB	’	7	G	W	g	w			ァ	キ	ヌ	ラ		
1000	BS	CAN	(	8	H	X	h	x			ィ	ク	ネ	リ		
1001	HT	EM	)	9	I	Y	i	y			ゥ	ケ	ノ	ル		
1010	LF	SUB	*	:	J	Z	j	z			ェ	コ	ハ	レ		
1011	VT	ESC	+	;	K	[	k	{			ォ	サ	ヒ	ロ		
1100	FF	FS	,	<	L	¥, \ †	l	|			ャ	シ	フ	ワ		
1101	CR	GS	−	=	M	]	m	}			ュ	ス	ヘ	ン		
1110	SO	RS	.	>	N	^	n	‾			ョ	セ	ホ	゛		
1111	SI	US	/	?	O	_	o	DEL			ッ	ソ	マ	゜		未定義

下位4 bit

ASCII keyboardではshift側

通常，shiftと関係ないkeyとする

† ASCII code

4.1.2 2進法の加算と減算

例題4.4 10進数の計算を2進数で行う

10進数の13と27の加算を2進法で行うためにはどうすればよいか考えよ．さらに，72から33を引く場合はどうすればよいか．

［解］

10進数の13と27の加算を2進法で行うと，**図4.4(a)**のようになります．すなわち，13は2進数で“1101”，27は“11011”で，その各桁の加算法則は**図4.4(b)**のように被加数Xと加数Yおよび下の桁からの桁上げ(Carry（キャリ）)cの三つの組み合わせからなります．

次の2進法の減算も，同じように行うことができます．10進数の72から33を引くことに相当する2進法の計算は，**図4.5(a)**のように加算と同じく下から順次計算すればよいのです．

ここで1から0は引けますが，0から1は引けないので，さらに上の桁から借ります．も

第 5 章

Flipflop, Registerと同期式論理回路

汎用論理ICである74 seriesで回路を構築していた時代は，回路の実装密度も低く，少ないICで多くの機能を実現する技術がもてはやされました．その結果flipflopやcounterも多くの種類が用意され，使い勝手に応じて利用されました．

GAやStandard cell，FPGAのようなLSIが主力になった今では，flipflopの動きが単純なほど設計ミスが少なくなり，検証も楽であるという認識で回路を作成します．Digital system設計者に対する評価では，いかに楽に検証できる回路を作るかという技術力が，処理速度を上げることと同じ程度に重要視され，回路数を減らすことにはあまり熱心にならなくてもよくなりました．

5.1　Flipflopの原理を理解する

前章までのgate回路では，入力端子の‘1’，‘0’(H，L)が決まれば，それに応じてある動作遅れ時間後に出力が一義的に決まりました．さらにgate回路をいくつか組み合わせてつないでも，動作遅れが増すのみで，やはり出力は一義的に決まります．

その入力の一つに自分自身の出力をfeedback(フィードバック)(帰還)して使うということを考えます．すると出力の値は，外部からの入力の値から一義的には決まらなくなります．それは現在の自分自身の出力も論理操作の対象となるからです．

これらの回路の動作を歴史的な進歩も考慮に入れて，原理からの理解を進めます．

5.1.1　論理回路にfeedbackを付ける

例題5.1　Feedbackのある回路

Feedbackのある回路を用いてdataの状態を保存するにはどうすればよいか．

［**解**］

ある論理回路で入力をA，B，C，…とし，出力をYとすると，その論理機能は一般的に，

$$Y=f(A,\ B,\ C,\ \cdots)$$

という論理式で記述できます．このような回路を組み合わせ論理回路(Combinational circuit(コンビネーショナル))と呼んでいます．組み合わせ論理回路では，回路のもつある遅延時間以上経過すれば，入力条件のみから出力が得られ，回路は論理式通りに動作したと見なせます．

図5.1(a)の回路と異なり，出力を表す論理式fの中に自分自身の出力Yが，

$$Y=f(Y,\ \cdots,\ A,\ B,\ C,\ \cdots)$$

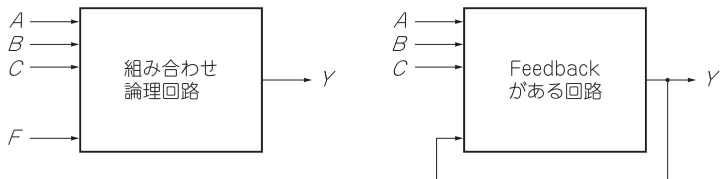


図5.1
Feedbackがある回路

(a)一定時間内に出力が一義的に決まる

(b)出力は内部状態で異なることがある

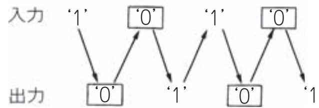
と入力論理変数として含まれている場合はどうでしょうか．これは**図**5.1(b)の図中に示すように，組み合わせ論理回路の出力が，入力へfeedbackしている回路になります．

Feedbackのもっとも簡単な形は，**図**5.2のようにinverterの出力を直接入力側につないだものです．この回路では，入力を'1'（H）とすると出力は'0'（L）になり，この出力の'0'（L）はfeedbackして入力を'0'（L）にします．すると，入力が'0'（L）なので出力は'1'（H）にならなければならず，結局，出力はこのinverterの遅延時間の2倍の周期で，'1'（H）と'0'（L）の間を交互に往復して発振することになります．

このように時刻$t+1$のときの出力Y_{t+1}が，時刻tのときの出力Y_tを含む入力条件によって，

$$Y_{t+1}=f(Y_t,\ A,\ B,\ \cdots)$$

というように決まる回路を順序回路（Sequential（シーケンショナル） circuit）といいます．



実際に発振させるには3段以上つないで，遅れ時間を確保する

図5.2
もっとも簡単な順序回路

ただし，**図**5.2の回路は立ち上がり時間t_rが遅延時間t_dとほぼ等しくなるために，普通は原理通りには発振しません．原理通りに動かすには，遅延時間を大きくしなくてはならないので，inverterを最低でも3段くらいつなぐ必要があります．このような回路をlogical oscillator（オシレータ）といい，回路の実質的な遅延時間の測定に用います．奇数個のinverterをつないだものは，すべて同じ発振動作となります．

では，次にinverterを偶数個つないだ場合を考えてみます．このもっとも簡単なものは，**図**5.3のように二つのinverterからなる場合です．この回路の二つのinverterは正論理から負論理，負論理から正論理に変換するinverter機能として使われています．したがって，左端が'1'のとき回路内のnode（ノード）（つなぎ目，節）はすべて'1'となり，また，左端が'0'のとき内部のnodeはすべて'0'となり二つの安定な状態が存在します．

もちろん，これをHやL levelで見た場合でも，左端がHならL，Hとなってfeedback

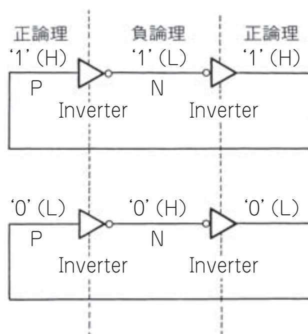


図5.3
2連inverter回路による基本flipflop
（P：Positive，N：Negative）

され，左端のHと一致しています．もう一つの場合も，左端がLなら順にH，Lとなり，やはり左端がLなので矛盾しません．このように，この回路は**図**5.2のlogical oscillatorと異なって，状態が安定していて発振現象は起きないことが分かります．

ところでこの回路で，最初に状態が‘1’，‘0’，‘1’となっていたり，逆に‘0’，‘1’，‘0’となっていると，どのようになるかを考えてみます．この状態は，回路として非常に不安定な状態ですが，通常は入力が出力に伝搬する速度が二つのinverterで差があるので，たちまちどちらかの安定状態に収まってしまいます．したがって，このような不安定な状態が続くことはないと考えてよいでしょう．

このように安定状態が二つある回路は，その安定状態の出力が‘1’か‘0’かによって，二者択一の情報の記憶が可能となります．一般にこのような回路をflipflop（フリップフロップ）と呼んでいて，digital回路でもっとも基本的な機能を実現する回路の一つです．

5.1.2　RS-flipflopの原理

例題5.2　RS-flipflopの動作とその解析
Flipflopを実際に利用するためにはどのような回路にすればよいか考えよ．

［**解**］

図5.3のflipflopは安定な状態が二つありますが，それを利用する場合，一方の状態から他方の状態への変化が可能であったり，あるいは指定の状態を外部からsetできないとdigital回路として利用価値が低くなってしまいます．それには**図**5.3のようなinverter回

路だけの組み合わせでは無理で，**図**5.4のように2入力以上の回路を用いてfeedback loopを外部から切れるようにしなければなりません．

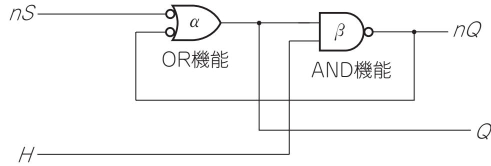
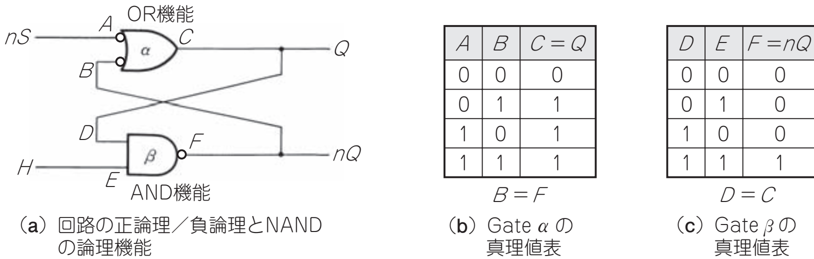


図5.4
NAND素子によるflipflop

この回路は二つのNAND素子で構成されていて，正論理，負論理が明らかになるように記述されています．この回路の一方の入力端子をset（セット）(*nS*)，もう一方をhold（ホールド）(*H*)と名付け，出力端子を*Q*，*nQ*とします．*n*が付いている信号は負論理で使われていることを意味しています．このflipflopがどう動くか調べてみます．



（a）回路の正論理／負論理とNANDの論理機能

A	B	C＝Q
0	0	0
0	1	1
1	0	1
1	1	1

B＝F

（b）Gate αの真理値表

D	E	F＝nQ
0	0	0
0	1	0
1	0	0
1	1	1

D＝C

（c）Gate βの真理値表

図5.5 Flipflopの論理機能

図5.4の回路を見やすくするために，**図**5.5(a)のように書き直します．この回路で使っている二つの素子は両方ともNAND素子ですが，αは入出力の正論理/負論理を考えるとORの機能となり，βはANDの機能となります．その真理値表は**図**5.5(b)，(c)のようになります．ここで入力*nS*を‘0’，*H*を‘1’としておき，出力*Q*，*nQ*がともに‘0’のときから以下のように調べてみます．なお，出力端子*nQ*は負論理なので注意してください．

(1) 初期状態では**図**5.6(a)のように，αの入力は二つとも‘0’なので，真理値表から出力は‘0’，βの入力は‘0’と‘1’でANDは成立せず，出力は‘0’となり，設定した*Q*，*nQ*の状態と一致しており，安定している．

(2) ここで入力*nS*を‘0’から‘1’にすると，αがORなので**図**5.6(b)のように*Q*は‘1’となり，βの二つの入力が‘1’，‘1’になる．その結果，ANDが成立して*nQ*も‘1’となる．これ

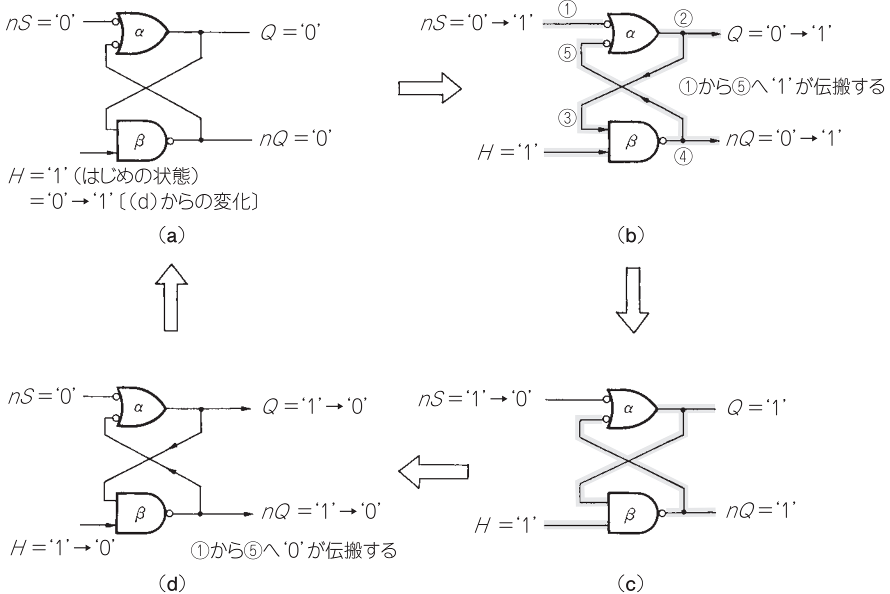


図5.6 Flipflopの動作

によってαのもう一方の入力が'0'から'1'へ変わるが，これはαがORなので，すでに出力Qは'1'に変わっているので，安定な状態となる．

(3) この状態でnS入力を'0'へ戻してみる．しかし，こうしても**図5.6**(c)のように，αの出力は'1'のまま変化しないので，flipflopの出力の状態は変わらない．

(4) 同じように，入力Hを'1'から'0'にすると，**図5.6**(d)のようにβのANDが成立しなくなり，出力nQは'0'となる．この結果，OR機能のαの出力Qも両方の入力が'0'なので'0'になる．出力Qが'0'になっても，βはすでに出力が'0'なので変わらない．この状態も安定な状態となる．

(5) ここで入力Hを'0'から'1'へ戻しても，βのANDは成立しないので，**図5.6**(a)と同じ状態に戻るだけで出力の変化はない．

この回路の動作をまとめると，nS = '0'，H = '1'が基準状態の入力条件です．

ここでQが'0'のとき，入力端子nSを'0'から'1'にする(nS = '1'，H = '1')と，出力Qは'1'となり，Qが'1'のときnSを'1'から'0'や，'0'から'1'に変えてもQは変化しません．

一方，Qが'1'のとき入力端子Hを'1'から'0'へ変えると(nS = '0'，H = '0')，Qは'0'

第 6 章

Digital Systemの構築
― State machineへの道 ―

Digital systemはdigital回路の延長線上にあります．しかし概念としては一つ上に位置しています．Computerの設計の世界では昔から必要な数のregisterを用意して，それからその間のdata移動や加工の論理を組み合わせ論理回路で書くことが当たり前でした．

この考え方が素直にRTL(Register Transfer Level)記述に変わったわけです．したがってregister類をどう用意するかは，digital systemをうまく設計する際の上手い下手に直接影響します．さらに，digital systemの構築にはstate machineの考え方も必要です．

6.1　Digital SystemのICへの組み込み

Digital systemは一つないし複数個のICの組み合わせで構築します．Digitalの実際の機能の多くはICの中で実現されています．

量産を行うときはGA(Gate Array)などに，実験や中小規模生産のときはFPGAやCPLD内に機能を組み込みます．主に汎用論理ICを使う時代は前世紀で終わりました．

6.1.1　自分で内容を書き込める論理IC

例題6.1　昔のdigital system

前世紀の90年代初頭まで74 seriesの汎用論理ICをprint基板に並べて，自分が必要とするdigital処理回路を作っていた．その歴史的な流れを追ってみて，大規模IC化の必然性を考えよ．

［**解**］

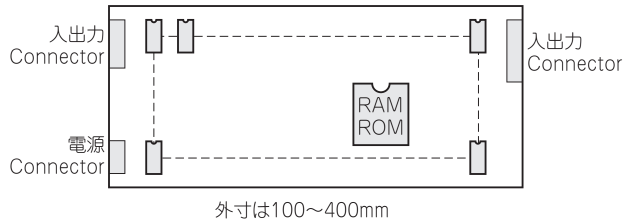


図6.1
すべて汎用論理ICを使ったdigital system

従来は**図6.1**に示すように，RAM/ROM以外は74 seriesに代表される汎用論理ICを使って，種々の論理機能の配置と配線はprint基板上で行って，digital systemを実現してきました．この場合必要によっては通常のprint基板でなくても，万能基板に手で配線をすれば実験用などの回路はたった1台でも作ることができました．

かつて「ミニコン」と呼ばれる誰でも使えるcomputerが出現した1960年代の終わりには，74 series ICを並べる方法でこの「ミニコン」が作られていました．1970年代に一世を風靡（ふうび）した「インベーダーゲーム」も初めはこのようにして作られていました．ただ，汎用ICを使う方法では，digital systemへの多大な要求に対処できなくなっていきました．

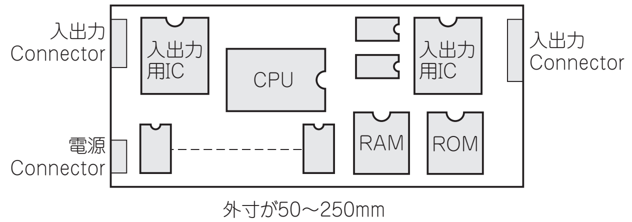


図6.2
高機能LSIと汎用論理ICを使ったdigital system

1970年代には演算機能を中心としたICであるCPUが市販されました．その後は，CPUなどの機能LSIと汎用論理ICの組み合わせで，**図6.2**のようにprint基板上にdigital systemを作るようになりました．これは大規模な汎用ICが用意されたことと同じであると解釈できます．

> **例題6.2　Field programmable logicの時代へ**
> 以前は実験や中小規模製品を設計するときに，74 seriesの汎用論理ICをprint基板に並べて，自分が必要とするdigital処理回路を作ってきた．FPGAが出現してdigital回路の作り方はどう変化したか．

［**解**］

● FPGA以前の設計

ICを作る技術がどんどん進み，いろいろな機能が一つのICの中に盛り込めるようになりました．**図6.3**のように，電卓や時計だけでなくいろいろの機器をたった一つのLSIだけで動かせるようになりました．かつて実際に汎用論理ICで回路を作っていると，いろいろな問題がありました．

(a) 価格が高くなる

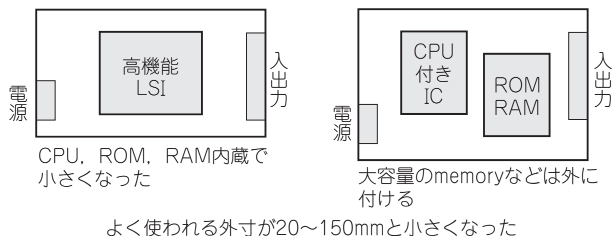


図6.3
高機能ICだけでdigital systemを作る

(b) 電源消費が大きい
(c) 外形寸法が大きい
(d) 速度が遅い
(e) 回路図を使うので設計の一覧性が悪い

などです．これらはどれも致命傷とも言える弱点です．しかし，汎用論理ICをなかなか捨てられなかったのは，

(f) MIL記号による簡便な設計
(g) 専用ICを作る時間と費用
(h) 動作検証の手段

などに満足できる解答がなかったからです．

高専や大学などにおけるdigital回路の教育にも問題があったかもしれません．学校教育はしばしば技術の世界の最先端から遅れる傾向があります．

現在は量産用ならgate arrayやstandard cell（スタンダード・セル）などが半導体各社から用意されていて，HDLを使ったり学んだりする人たちの多くが，この水準の設計に従事しています．

● FPGAが設計の世界を変えた

IC製造技術の発展によりFPGA（Field Programmable（フィールド・プログラマブル） Gate Array）やCPLD（Complexed（コンプレックスト） Programmable Logic Device）などの，一つずつ内部の論理回路を外部から書き込むことができるICが出現しました．なお，これらの内部構造については参考図書やdata sheetを参照してください．

研究室で1台だけとか，工場で100台だけとかのdigital systemも，従来の74 series汎用論理ICを使っていた時代のような「高い・遅い・でかい」を我慢せずに，FPGA/CPLDを使って大規模ICの恩恵に預かることができるようになったのです．

歴史は戻りますが，74 seriesではなくても任意の論理回路を作りたいという考えを実現したICとしてPAL（Programmable Array Logic），GAL（Generic（ジェネリック） Array Logic）など，userが内部の論理を自由に書き込める論理ICが作られました．

PLD（Programmable Logic Device）というのはこれらのIC類の総称です．PALは一度回路を書き込んでしまうと内部の構成は消せないので，あまり流行りませんでした．GALはEEPROMの技術を使っていて，書き込んだ回路を消すことができるので，まだ現役で使うこともあります．

現在本書で設計のtarget ICとして考えているのは，GALが発展した形のCPLDやGA

と同じ発想で回路が作成されるFPGAです.

例題6.3 FPGA/CPLD時代の設計

FPGAやCPLDを使って自分が必要とするdigital処理回路を作るには，どんな手順でやればよいか.

［**解**］

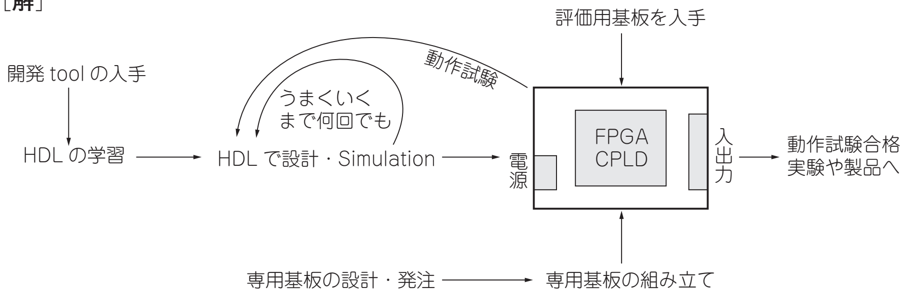


図6.4 FPGA/CPLDで独自のdigital systemを設計する手順

FPGAやCPLDでdigital systemを設計する手順は，おおよそ**図6.4**にある通りです.以下，順を追って説明します.

(1) Digital回路を組み込む対象となるvendor(Altera社やXilinx社など)の開発toolを購入するか，無償の機能限定版をwebsiteからdownloadで入手する．Softwareのsizeは非常に大きく1 GByteを越えることもあるので，HDD(Hard Disk Drive：ハードディスク)のC driveの空き容量に注意する.

(2) Target(ターゲット)のFPGA/CPLDが付いているprint基板を用意しておく．このときFPGA/CPLDのprogramに必要なcable(ケーブル)類も用意する．CableはパソコンのUSB portやprinter portからJTAGあるいは特別なconnector(コネクタ)に接続するものである．Vendorの純正品以外にも安価な代替品(コンパチ品ともいう)がある.

(3) Print基板は，回路が比較的簡単なときは，市販されているFPGA/CPLDの評価用基板を使うのがよいだろう．回路が複雑であったり，基板形状や機能に独自性が必要なときはprint基板を特注する.

(4) 使用予定のFPGA/CPLDの回路数が，設計する回路規模と比べて数倍のゆとりがある場合は，FPGA/CPLDへの入出力信号やdataのpin(ピン)(端子)配置を適当に決めて，print基板を先行して作成する.

(5) 基板を先行作成できないときは，汎用の基板で実験を進めて，ある程度の目処が立ってからprint基板を作成する．そうしないと指定した信号をICの指定pinへ配置することが困難になることがある．

(6) 論理回路を仕様書通りに設計する．仕様書はかならず作っておく．きちっと書かれた仕様書がないと，後で変更するときなどに困る．

(7) 設計言語には，VHDL，Verilog HDL，回路図などが使える．本書では，Verilog HDLを用いている．回路図入力は以前の74 series汎用IC時代から設計をしていた技術者のための方式で，HDLがよく理解できない人の救済策だと考えてほしい．

(8) HDL自体の形式や設計のやり方および開発toolの使い方などについては，各toolのhelp，websiteの情報およびHDL専門図書で確認する．

(9) 設計した論理回路はtool上で動作をsimulationしてから，FPGA/CPLDに書き込む．Simulationをしていないと，FPGA/CPLDの内部の信号は測定器では追いかけることができないので，動かないときは何がなんだか分からなくなる．

(10) 実回路で期待した動作が検証できればHDLによる設計は完成である．だめなら，設計に戻ってやり直しになる．

Tool類は自己完結しているので，簡単に自分の希望のdigital systemを作成できます．

さらに，CPUを含めてPCI busのinterface回路などいろいろな機能回路が，設計済みの回路(IP：Intellectual Property（インテレクチャル・プロパティ）)としてFPGA/CPLD vendorから提供されているので，それらについては自分でtimingなどを調べる必要がありません．この点では，FPGA/CPLDは独立機能の大規模ICを組み合わせる**図6.3**と同じことができます．

6.1.2 注文生産のdigital IC

例題6.4 注文生産のICの作り方

携帯電話や時計，gameなど情報機器に組み込まれているdigital ICはどのように作られるのか，簡単に説明せよ．

［**解**］

注文生産の大規模なIC(以下LSIのことを意味する)の作り方には，大きく分けて3種あります．

- すべての回路の半導体patternを零から(From scratch（スクラッチ）などという)起こすfull custom（カスタム）.