第1章

基本論理と論理回路を表現する方法

Digital 回路の基本は“0”と“1”および AND，OR，NOT の論理です．この章ではこれら歴史的な感じがする事項を，HDL で設計するという観点から整理します．

すでに多くの読者はこのあたりのことはご存じかもしれませんが，本書の記述法の特徴をご理解いただくためにも，ざっとは一読してください．できれば例題をやってみて，Tool 類の使い方に習熟してください．

1.1　論理の表現法

1.1.1　“ 0 ”と“ 1 ”

例題 1.1　2 値論理

2 値論理について考えよ.

［**考え方**］

Digital は世の中のすべてのことを“ 0 ”と“ 1 ”の二つの値（論理値という）で表現しようとする考え方の世界です.

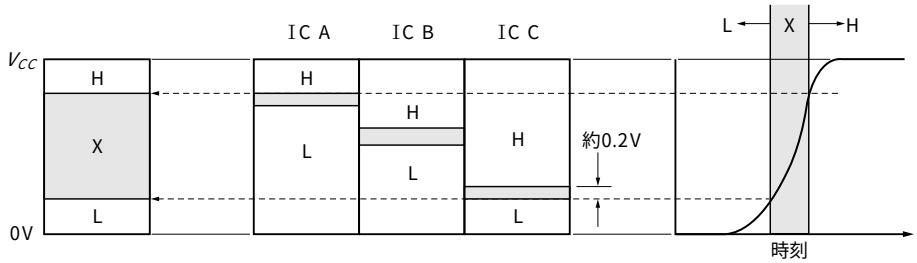
もちろん多くの事柄を表すには，たった一つの Digital の論理値だけでは不十分なので，たくさんの Digital の論理値を組み合わせて表現します．CD も DVD もそのような考えで作られているので，Analog 値が表現できているように見せることができるのです.

［**解**］

論理回路はこの“ 0 ”と“ 1 ”の二つの値しかとらない 2 値変数（Boolean という）の演算を物理的に実現する方法の一つです．この場合，物理的ということは電気的ということです.

電気信号としては**図 1.1** に示すように，電位を基にして論理状態を記述します．電圧が高い状態を“ H ”，低い状態を“ L ”と定義します．その中間の電位は閾値（いきち，しきいち： Threshold level）を境に，H か L のどちらかであると判定します.

〔図 1.1〕
電気信号と
Threshold level



［**注意点**］

H と L の境界は電源電圧にもよりますが，伝統的な TTL compatible interface では，各回路ごとに 1.4 V 付近で 0.2 V ほどの境界領域の幅があります.

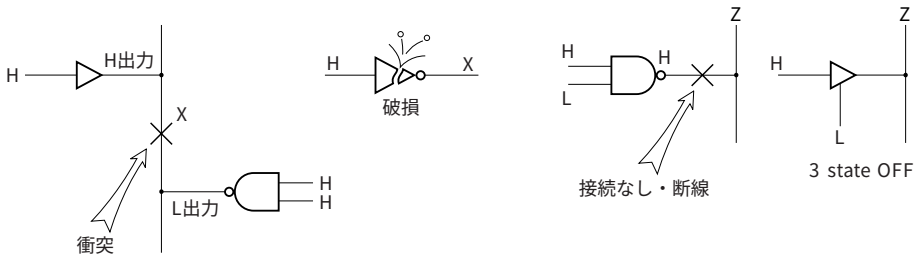
境界の値は**図 1.1** のように同じ IC の内部でも各論理回路で負荷などの原因からまちまちになり，全体としても使用温度範囲や電源電圧範囲の幅も見て，System での境界領域の許容値を 0.8 V 〜 2.0 V などと決めます.

System で決めた許容値幅の間に現実の電圧があるときは，H でも L でもない状態 X と考えます．X という状態が発生する原因としては，電気信号が L と H の間を遷移している状態のときだけではなく，**図 1.2** の回路に示すように，H と L の出力が衝突しているときや単にその出力回路が壊れている場合もあります．

設計の段階ではさらに，回路を動かす前でまだ H とも L ともわからない状態も X で表現します．この X 状態は回路設計では基本的には考えなくてもよいのですが，設計した回路を検証する際には問題となります．

このほか，**図 1.2** のように H と L のどれにも接続されていない Z という状態も考えられます．Z の状態は後述する 3 state 論理回路として Bus 接続の Data などにはよく使われます．

〔図 1.2〕
X，Z 状態



（**a**）X 状態の原因　　（**b**）Z 状態の原因

1.1.2　正論理と負論理

例題 1.2　負論理信号の命名

入力信号 Inp1 と Inp2 を負論理として使いたい．適当な信号名を考えよ．

［**考え方**］

電圧が高い状態 H のとき，Data や制御信号，状態の検出信号が有効（有用）であるとする立場を正論理（H と“ 1 ”が対応）といい，L のとき同じく有効とする考え方を負論理（L と“ 1 ”が対応）といいます．

負論理は，多くの IC では信号 Pin のところに**図 1.2** のように○印を付けて，IC の利用者の注意を喚起していますが，これは MIL 記号法の名残りです．このような表記がなされていない IC もあるので，制御信号などはよく調べてから使うようにします．

多くの HDL simulator や Compiler では負論理をとくに意識せずに，H はすべて“ 1 ”と書く IC 製造者の習慣をそのまま採用しています．したがってある信号が“ 0 ”と表現されていても，それを受け取った側ではなにか有用な動作をしなければならない場合があります．

負論理の信号であることを明確にするために，以下のように信号の名称に負論理を示す記号や文字を加えることが，Digital 技術の普及以前から採用されてきました．

nSignal ，NSIG ，SIGN ，SIG/，sig#，－SIG ，S̄īḡn̄āl̄

本書では外部とやりとりする信号名は，原則として大文字で始めます．負論理は小文字の n をさらに頭に付け加える方法で意図的に表現することにします．

［**解**］

本書での記述法の原則から，信号名の前に n を付けて，nInp1，nInp2 とします．

［**注意点**］

本書の説明文では論理値の“0”と“1”は，“ と ”でくくってわかりやすいようにしています．

Verilog-HDL では 2 値の数値をそれぞれ**表 1.1** のように表現します．1 bit 幅の Binary（2 進）data の“1”という意味です．単純に 0，1 としていないのは，10 進数の値の 0 や 1 と区別するためです．Verilog-HDL の処理系によっては，supply1 や supply0 が使えることもあります．

AHDL の論理式表現では，同表のように“0”を供給する際は GND と書き，“1”は VCC と書きます．

Verilog-HDL では大文字と小文字を区別するので，通常の Windows の File 名や Program のときのように，大文字と小文字を適当に書いてはいけません．

一方，**AHDL では大文字小文字の区別をしない**ので，GND や VCC は Gnd や Vcc と書いても同じです．

〔表 1.1〕
正負論理と“0”と“1”の表現方法

論理値	正論理	負論理
“0”	L	H
“1”	H	L

電位	Verilog-HDL の 2 進値	AHDL
L	1'b0	GND，gnd
H	1'b1	VCC，vcc

例題 1.3　論理値“1”の供給（AHDL の論理式）

信号 A と信号 nB が宣言されていて，それぞれ正論理と負論理の信号として使うとき，論理値の“1”を供給する式を AHDL の論理式で書け．

［**考え方**］

HDL では，Data や信号は**図 1.3** のように，設計している回路を一つの Block として見たとき，その Block への Input と Block からの Output および内部で使われている Data や信号（soft，node，wire などと名付けられている）として宣言します．

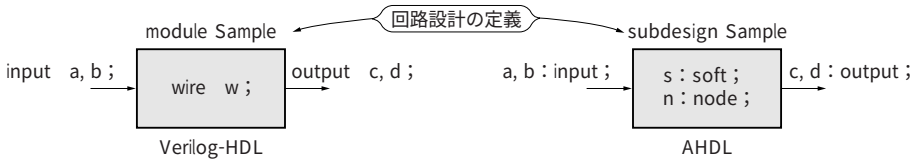
表 1.1 から負論理では，Ground 電位を供給すると論理値“1”が供給されたことになります．

［**解**］

AHDL の論理式の表現は，つぎのようになります．

```
A  = Vcc;
nB = Gnd;
```

〔図 1.3〕
信号の定義



[**注意点**]

完全な論理設計には，回路 Block 名の定義・信号名の定義・論理実現部の区切りなどが必要ですが，ここでは省略します．

AHDL の論理式や Verilog-HDL では式の終わりは Semi-colon 記号 ; で表します．

課題 1.1　論理値‘1’の供給（Verilog-HDL の論理式）

信号 C と nD がそれぞれ宣言されていて，正論理と負論理の信号として使うとき，論理値‘1’を供給する式を Verilog-HDL の論理式で書け．

[**考え方**]

Verilog-HDL では信号への代入は，AHDL の論理式のように＝記号だけからなる式ではなく assign 文の記述部として，

```
assign A = B;
```

のように書きます．

[**注意点**]

Verilog-HDL では 2 進数の値 1'b1 でなくとも，10 進数の値として 1 を書いても，Data 幅が 1 bit のときは同じことを意味します．

1.2　論理機能の表現

1.2.1　AND，OR と NOT

例題 1.4　AND，OR と NOT 機能の真理値表

AND，OR，NOT という論理機能を真理値表（Truth table という）を用いて説明せよ．

[**解**]

表 1.2 に示すように，以下のような定義ができます．

AND は二つの入力が両方とも‘1’のときだけ出力が‘1’となるような論理機能

OR は二つの入力の最低どちらか一つが‘1’のとき出力が‘1’となるような論理機能

NOT は入力が‘1’のとき出力が‘0’，逆に入力が‘0’のとき出力が‘1’となる論理機能

〔表 1.2〕
AND，OR，NOT の論理機能の真理値表

信号		状態			
入力	a	0	0	1	1
	b	0	1	1	0
出力	y	0	0	1	0

（**a**）AND の論理

信号		状態			
入力	a	0	0	1	1
	b	0	1	1	0
出力	y	0	1	1	1

（**b**）OR の論理

信号		状態	
入力	a	0	1
出力	y	1	0

（**c**）NOT の論理

［**考え方**］

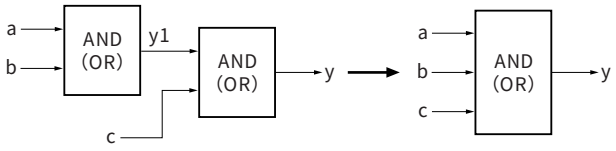
2値で表現できる世界では，このよく知られたAND，OR，NOTの三つの論理機能を使った組み合わせで，すべての状況を表現できます．こう言ってしまえば実も蓋もないのですが，現実はそう甘くはありません．

これら三つの論理機能を組み合わせてDigital回路で必要なすべての機能を実現していく過程は，限られた種類の煉瓦を組み合わせて建物を建てることと似ています．

ANDとORは2入力の場合しか表になっていませんが，三つ目以上の入力cを入れたいときは，**図1.4**のようにもう一つの2入力のANDまたはORを今の出力側につないで，余った入力に入れてやれば全体としてANDやORが取れます．

2入力の論理機能を多段接続してすべての論理機能を実現するのは，数学的には美しいかも知れませんが，遅れ時間などの問題で実用的ではありません．実際には**図1.4**の右の図のように多入力の論理機能を用意して使います．

〔図1.4〕
多段接続による入力数の拡大



［**注意点**］

この真理値表は，入力信号の組み合わせが“0，1”の次が“1，1”となっています．

これは隣り合う状態の間では，入力信号のうちただ一つだけが同時に変化するように並べたからです．表の最後は表の頭に繋がっていると考えます．

このようにある状態から別の状態へ移るには，いくつの信号の“1”，“0”が変わればよいかという数を，その状態間の論理（Bool）空間での距離ということがあります．この場合は距離1で並べたというわけです．

1.2.2　論理機能を表現する論理式

例題1.5　論理機能の表現

論理機能を真理値表以外の方法で表すことを考えよ．

［**考え方**］

論理機能は，入力の数が有限なら，そのすべての場合の組み合わせを用意し，それに対しての出力をすべて前節の真理値表のように書き出せば，原理的には完全に記述できます．

これは九九の表のようなものですから，入力の数が増えるとぱっと見て論理機能を簡単に理解することはできません．しかし他の手段が尽きたときの究極の表現方法です．

まず考えつく方法は，論理機能を表現する式（論理式）を導入することです．さらに図で表すことも有力な手段です．これは後の節で考えます．

［**解**］

Bool 代数での表現

論理機能を表現できる数学があります．Bool 代数と言われる，集合論の考えを利用したものです．集合とはある条件下のもの（要素）の集まりです．

集合論では，ある集合を構成する要素が一つもない集合を空集合（ϕ）といいます．また，その集合で考えている宇宙を構成する要素すべてを含む集合を，全体集合（R）と言います．要素を部分的に含む集合をこの全体集合の部分集合と言います．ϕ と R も部分集合に含めます．

たった一つの要素しかない宇宙での，その要素を含む全体集合を“ 1 ”，含まない空集合を“ 0 ”と記号を付けると，集合論の演算式が論理式になります．

●共通集合（AND 機能）

（∩はこの記号の両側の部分集合の共通要素だけを取り出して，新しい部分集合にする演算子）

$\phi \cap \phi = \phi$　→ “ 0 ” and “ 0 ”=“ 0 ”;

$\phi \cap R = \phi$　→ “ 0 ” and “ 1 ”=“ 0 ”;

$R \cap \phi = \phi$　→ “ 1 ” and “ 0 ”=“ 0 ”;

$R \cap R = R$　→ “ 1 ” and “ 1 ”=“ 1 ”;

●和集合（OR 機能）

（∪はこの記号の両側の部分集合のすべての要素を取り出して，新しい部分集合にする演算子）

$\phi \cup \phi = \phi$　→ “ 0 ” or “ 0 ”=“ 0 ”;

$\phi \cup R = R$　→ “ 0 ” or “ 1 ”=“ 1 ”;

$R \cup \phi = R$　→ “ 1 ” or “ 0 ”=“ 1 ”;

$R \cup R = R$　→ “ 1 ” or “ 1 ”=“ 1 ”;

●補集合（NOT 機能）

（C はこの記号が付いている部分集合の要素でない要素を，全体集合 R から取り出して新しい部分集合にする演算子）

$C(\phi) = R$　→ not (“ 0 ”) = “ 1 ”;

$C(R) = \phi$　→ not (“ 1 ”) = “ 0 ”;

●集合論で成り立つ法則類

（A ，B ，C は部分集合および “ 0 ” と “ 1 ” の 2 値だけを取る変数（Boolean））

結合則　$A \cap (B \cap C) = (A \cap B) \cap C$　→ A and (B and C) = (A and B) and C ;

　　$A \cup (B \cup C) = (A \cup B) \cup C$　→ A or (B or C) = (A or B) or C ;

交換則　$A \cap B = B \cap A$　→ A and B = B and A ;

　　$A \cup B = B \cup A$　→ A or B = B or A ;

分配則　$A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$　→ A and (B or C) = (A and B) or (A and C);

　　$A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$　→ A or (B and C) = (A or B) and (A or C);

●Bool 代数の演算子の記号

（Bool 代数の演算子に，集合論の演算子に似た記号や代数の記号を使うこともある）

and →∧，・; or →∨，＋; not (A) → $\bar{A}$ ，¬ A

●演算子の優先順位

not，and，or の演算子は，もしカッコなどで優先順位が指定されていない場合は，この順に優先的に演算が行われます．

課題 1.2　論理式の演算

Bool 代数で書かれた下記の論理式について，すべての場合を調べることにより式が成り立つことを調べよ．ただし A ，B は“0”と“1”の2値だけを取る変数（Boolean）である．

(a) not (not (A)) = A ;　　(f) A and not (A) =“0”;　排他律
(b) A and “1”　= A ;　　(g) A or not (A)　=“1”;　排中律
(c) A and “0”　=“0”;　　(h) not (A and B) = not (A) or not (B) ;
(d) A or “0”　= A ;　　(i) not (A or B)　= not (A) and not (B) ;
(e) A or “1”　=“1”;　　(h)，(i) は DeMorgan の定理

［**考え方**］

補集合の定義から not（not（“1”））=“1”と not（not（“0”））=“0”が成り立つことに注意してください．

例題 1.6　HDL での記述

Verilog-HDL や AHDL の論理式ではこれら AND，OR，NOT の機能をどう表現しているか調べよ．

［**解**］

Verilog-HDL と AHDL での記法を**表 1.3** に示します．

〔表 1.3〕
HDL で AND，OR，NOT などを表す記号
（y，a，b は変数）

論理	and	or	not
演算子	a & b	a | b	~a
論理機能	and(y, a, b)	or(y, a, b)	not(y, a)

(**a**) Verilog-HDL の記法

論理	and	or	not
演算子	a & b	a # b	!a
の記法	a and b	a or b	not a

(**b**) AHDL の記法

［**注意点**］

Verilog-HDL は C 言語の論理演算子を利用しています．これらの演算子は複数 Bit 幅の Data の各 Bit ごとに演算を実行する場合のもので，1 bit の Boolean については，&&（AND），||（OR），!（NOT）を演算子として使います．本書では実質的に同じことができるので，これらの記号は使わずに複数 Bit 用の記号を 1 bit data としての Boolean にも適用します．

Verilog-HDL の下の段は，基本論理機能（Primitive）を直接記述する方法です．論理記号化されていない機能を記述するにはこの方法しかありません．

まえがきで述べたように AHDL の論理式では歴史的な PAL（Programmable Array Logic）用の記述言語 CUPL 系列の記号を使っています．素直な表記法として and ，or ，not も使えますが，式が長くなるのと見にくいので使わないほうがよいでしょう．

課題 1.3　DeMorgan の定理

課題 1.2 の(h)，(i)に示す式を，Verilog-HDL および AHDL の論理式で記述せよ.

[**考え方**]

単純に記号を置き換えればよいのです.

1.2.3　MIL 記号法による表現

例題 1.7　論理機能の MIL 記号による回路図化

AND，OR，NOT の論理記号を MIL 記号で表せ.

[**考え方**]

論理機能に限らず，あらゆる接続関係(ある Block の出力が他の Block の入力になるような構造)にある事柄を示すには，旧約聖書のはじめに延々と書いてあるような文章による記述よりは，家系図のように図示したほうが直感的に理解できます.

論理機能も同じで，**図 1.4** に示したような形で書くとその上の部分に書いてある説明よりは簡単に理解できます.

ただ論理機能をすべて同じ四角の枠で囲って，その中に論理機能を文字で書いたのでは，いちいちその字を読んで回路を理解しなければなりません.

記号自体に AND や OR，NOT の意味をもたせ，パソコンの Display 上の Icon のようにすれば論理機能を書き込む手間もいらず便利です.

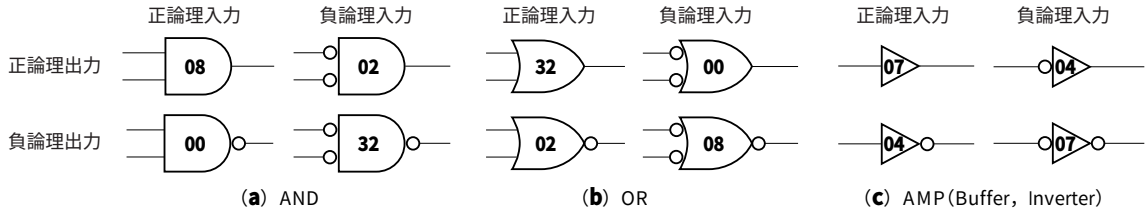
論理回路の設計を行っている所では，古くから自分勝手に記号を作って使っていました.

[**解**]

各者各様では図面を読まされるほうがたいへんなので，米軍が自分のところに納入される機器の図面は米軍が定めた方法で書くように制定したものが，**図 1.5** に示す通称 MIL 記号です.

この記号法の書き方や使い方の説明は，参考図書[1]を見てください.

HDL の一つである VHDL も，同様の趣旨で米軍が定めた記述法です.



〔図 1.5〕論理機能の MIL 記号による表記

［**注意点**］

MIL 記号法では，NOT を実現する物理的な論理素子は与えられていません．

1.1.2 節で説明したように，同じ信号が正論理と負論理の間を動くと NOT の動作となる，という立場です．これは昨今のような HDL 主体の論理設計に慣れた人にとっては解りにくい考え方です．

端的に言うと**図 1.5** の中の信号の入出力端に付けた○印が，信号線の両側で対応していないと，NOT の論理機能が自動で構成されるのです．

単純に NOT 機能を実現するには**図 1.5** の三角形で表された Amplifler 機能に○印が付いた Inverting buffer を使います．このときは，正論理から正論理への NOT，負論理から負論理への NOT 機能となります．

1.3　Data や状態を保持する機能

1.3.1　Register と Flipflop の機能

例題 1.8　Register の概念

Data を保持できる機能を考えよ．

［**考え方**］

Digital system を Digital 回路として物理的に実現するには，Data や信号の状態を保持できる機能をもたないと不可能です．

たとえば，室内にいる人数を数えるという Digital system の機能を実現するには，**図 1.6** のように今いる人数に入ってきた人数を加えて，出て行った人数を引かなければなりません．加減算の機能はいままで議論してきた AND や OR，NOT 機能の組み合わせで実現できます．

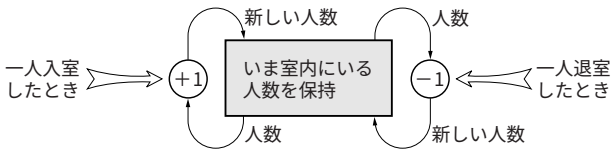
この機能の実現のためには，現在の人数という Data を保持する機能が必要です．このように Data を保持できる機能を Register と言います．

Register は電源が入っているかぎり Data を無限に保持できると考えます．

Register に保持させる Data をいつ Register に Set するかという信号が必要です．

もちろん最初に Register に保持されている Data の値を 0 などの指定した値にする機能も必要です．

〔図 1.6〕
室内の人数を数える Digital system



［**解**］

図 1.7 に Data を保持する機能として Register に要求される機能を付けた状況を Block 図で示します．