

# 計算システム論 1

Ver. 0.17.0

O-green

2017 年 6 月 16 日

# 目次

第 0 章	はじめに	1
第 1 章	ブール代数	2
1.0	公理集	2
1.1	基本概念	2
1.2	ブール代数の基本定理と公理系	3
1.3	順序関係の導入	4
第 2 章	論理関数の表現	6
2.1	論理関数と組み合わせ回路	6
2.2	論理式の標準展開	7
2.3	BDD 表現	9
第 3 章	論理関数の性質	10
3.1	ユネイト関数と単調関数	10
3.2	双対関数	10
3.3	その他の関数	11
第 4 章	論理合成	13
4.1	AND-OR 2 段形式	13
4.2	カルノー図による論理式簡単化	13
第 5 章	順序回路	16
5.1	オートマトン順序回路	16
5.2	順序回路の実現	17
5.3	フリップフロップ	17
5.4	順序回路の等価性	20
第 6 章	演算回路	21
6.1	データ表現	21
参考文献		23

## 第 0 章

# はじめに

授業の目的ハードウェアの設計に関する基礎理論 VLSI 上に実装する際の基本戦略高速 安い 信頼性

## 第 1 章

# ブール代数

### 1.0 公理集

$$\text{交換律} \begin{cases} x \vee y = y \vee x \\ x \cdot y = y \cdot x \end{cases} \quad (1.1)$$

$$\text{結合律} \begin{cases} (x \vee y) \vee z = x \vee (y \vee z) \\ (x \cdot y) \cdot z = x \cdot (y \cdot z) \end{cases} \quad (1.2)$$

$$\text{吸収律} \begin{cases} x \vee (x \cdot y) = x \\ x \cdot (x \vee y) = x \end{cases} \quad (1.3)$$

$$\text{分配律} \begin{cases} x \vee (y \cdot z) = (x \vee y) \cdot (x \vee z) \\ x \cdot (y \vee z) = (x \cdot y) \vee (x \cdot z) \end{cases} \quad (1.4)$$

$$\text{単位元, 零元} \begin{cases} x \vee 1 = 1 \\ x \cdot 0 = 0 \end{cases} \quad (1.5)$$

$$\text{補元律} \begin{cases} x \vee \bar{x} = 1 \\ x \cdot \bar{x} = 0 \end{cases} \quad (1.6)$$

$$\text{べき等律} \begin{cases} x \vee x = x \\ x \cdot x = x \end{cases} \quad (1.7)$$

### 1.1 基本概念

#### 1.1.1 束 (Lattice)

集合  $L$  上で 2 つの 2 項演算

$$\vee : L \times L \rightarrow L$$

$$\cdot : L \times L \rightarrow L$$

が定義され  $L$  の任意の要素  $x, y, z$  に対して (1.1), (1.2), (1.3), (1.7) の 4 つの公理が満たされる時この代数系  $(L, \vee, \cdot)$  を束という。

束  $(L, \vee, \cdot)$  において (1.4) が成立するとき分配束という。

束において任意の要素が補元をもつとき相補束という。補元は (1.5) で定義、単位元と零元を持つ時定義可能。

$$a \text{ の補元が } b \Leftrightarrow a \vee b = 1, a \cdot b = 0$$

相補的かつ分配的な束をブール束という。ブール束によって定義される代数系をブール代数という。

定理 分配束において要素が補元をもつときは補元は一意である。 ■

### 1.1.2 ブール代数の定義

定義 1.1(ブール代数)  $B$  を少なくとも 2 つの異なる要素 0 と 1 を含む集合とする。 $B$  の上で 2 つの 2 項演算  $\vee, \cdot$  と 1 つの単項演算  $\overline{\phantom{x}}$  が定義され (1.1) ~ (1.6) の 6 つの等式 (公理) を満たす時  $(B, \vee, \cdot, \overline{\phantom{x}}, 0, 1)$  をブール代数という。

ブール代数における演算

$\vee$  : 論理和, OR

$\cdot$  : 論理積, AND

$\overline{\phantom{x}}$  : 否定, NOT

$B$  の要素を値とする変数をブール変数 あるいは 論理変数 という。

ブール変数と定数に演算を 0 回以上適用したものをブール式 あるいは 論理式という。

演算の優先順位は以下のとおりである

括弧  $()$  > 否定  $\overline{\phantom{x}}$  > 論理積  $\cdot$  > 論理和  $\vee$

### 1.1.3 双対性

ある等式とその等式に対し  $\vee \leftrightarrow \cdot, 0 \leftrightarrow 1$  の交換をした等式は双対 (dual) であるという。

双対性原理ある等式が成立する時、その双対な等式も成立する。ただし演算の優先順位は保つ。

## 1.2 ブール代数の基本定理と公理系

### 1.2.1 基本定理

定理 1.2 ブール代数では以下が成立する。

$$\begin{aligned}
 (1.8) \quad & \begin{cases} x \vee 0 = x \\ x \cdot 1 = x \end{cases} \\
 \text{単位元、零元} \quad & \begin{cases} x \vee y = 0 \Leftrightarrow x = y = 0 \\ x \cdot y = 1 \Leftrightarrow x = y = 1 \end{cases} \\
 \text{ド・モルガン律} \quad & \begin{cases} \overline{(x \vee y)} = \overline{x} \cdot \overline{y} \\ \overline{(x \cdot y)} = \overline{x} \vee \overline{y} \end{cases} \\
 \text{二重否定} \quad & \overline{\overline{x}} = x \\
 \text{ブール吸収律} \quad & \begin{cases} x \vee (\overline{x} \cdot y) = x \vee y \\ x \cdot (\overline{x} \vee y) = x \cdot y \end{cases}
 \end{aligned}$$

定理 (1.3) 論理式  $F$  の否定  $\overline{F}$  は  $F$  において演算の置換 ( $\vee \leftrightarrow \cdot$ )、変数の置換 ( $x_i \leftrightarrow \overline{x_i}$ )、定数の置換 ( $0 \leftrightarrow 1$ ) を施して得られる論理式と等しい。

### 1.2.2 ハンティントンの公理

ブール代数の定理を証明するには (1.1), (1.4), (1.6), (1.8) の 4 つの等式で十分。

### 1.3 順序関係の導入

#### 順序関係

$S_1$  と  $S_2$  を集合とする。直積  $S_1 \times S_2$  の部分集合  $R$  を  $S_1$  と  $S_2$  の 2 項関係という。任意の要素  $s_1 \in S_1, s_2 \in S_2$  に対し  $(s_1, s_2) \in R$  のとき  $s_1$  と  $s_2$  は関係  $R$  をもつといい、 $s_1 R s_2$  と表記する。特に  $S_1$  と  $S_2$  が同じ集合の時  $R$  を  $S$  の上の 2 項関係という。

集合  $S$  の上の 2 項関係を  $R$  とする。 $S$  の任意の要素  $x, y, z$  に対して

$$\begin{cases} \text{反射率: } xRx \\ \text{反対称律: } xRy \wedge yRx \Rightarrow x = y \\ \text{推移律: } xRy \wedge yRz \Rightarrow xRz \end{cases}$$

が成立する時  $R$  を順序関係、あるいは半順序関係という。また、 $S$  のすべての要素に対して  $xRy$  あるいは  $yRx$  が成り立つような順序関係  $R$  を全順序関係という。

関係  $R$  を  $xRy$  と書く代わりに  $x \leq y$  と書くとわかりやすいので、以下この表記を採用する。

#### ハッセ図

順序関係を表す図としてハッセ図がある。集合  $S$  の異なる  $x, y$  に対し

$$\begin{cases} x \leq y \text{ であり} \\ x \leq z \leq y \text{ となる } z (z \neq y, x) \text{ が存在しない} \end{cases}$$

ときに  $y$  を  $x$  の上方に置き、両者を線で結んだものがハッセ図である。

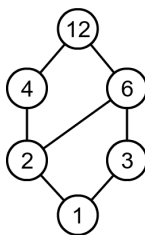


図 1.1  $x \leq y \Leftrightarrow$  「 $y$  は  $x$  で割り切れる」と定義した時のハッセ図

#### 最大、極大、上界、上限

半順序集合  $S$  のある要素  $s$  に対し  $s \leq a$  となる  $a$  が常に  $s = a$  となるとき  $s$  を  $S$  の 極大元 という。

半順序集合  $S$  のある要素  $s$  が  $S$  の任意の要素  $a$  に対して  $a \leq s$  を満たすときの  $s$  を  $S$  の 最大限 という。

極小元や極大元も同様に定義される。

最大限や最小限はたかだか 1 つであり、ない場合もある。最大限 (最小元) は存在するならば必ず極大元 (極小元) である。

関係  $\leq$  が定義された半順序集合  $S$  の部分集合を  $T$  とする。

$T$  の任意の要素  $t$  に対して  $t \leq u$  が成立する  $S$  の要素  $u$  を  $T$  の 上界 という。

上界全体からなる集合の最小元を上限という。

下界、下限も同様に定義される。

## 束

半順序集合  $(L, \leq)$  において  $L$  の任意の 2 つの要素が必ず上限と下限をもつときその半順序集合を束という。

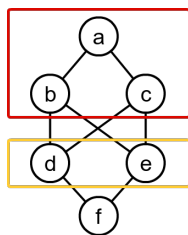


図 1.2  $d, e$  の上界は  $a, b, c$  であるが、上限は存在しない。この順序集合は束ではない。

束の順序構造としての定義と、代数的定義 (1.1) は等価である。順序構造における代数的演算として

- $x \vee y$  を  $x$  と  $y$  の上限
- $x \cdot y$  を  $x$  と  $y$  の下限

と定義すれば、(1.1) での代数的定義と矛盾しない。

## 束における双対性

束である順序集合  $A$  において、与えられた任意の正しい命題は、関係「 $\leq$ 」「 $\geq$ 」を交換、演算「 $\vee$ 」「 $\cdot$ 」を交換しても正しい命題となる。これらの交換は、ただ単にハッセ図を上下ひっくり返す操作に対応する。

## 種々の束

分配律が成立する束を 分配束 と呼ぶ。零元と単位元が存在し、任意の要素に対し補元が存在する束を 相補束 と呼ぶ。相補的かつ分配的な束を ブール束 という。

## 第 2 章

# 論理関数の表現

## 2.1 論理関数と組み合わせ回路

### 2.1.1 論理関数

#### 論理回路

AND, OR などの論理素子の相互接続で構成される回路を論理回路という。一般には複数の入力と複数の出力を持つ現在の出力が現在の入力のみで決定されるような論理回路を組み合わせ回路 (combinational circuit) と呼ぶ。それに対して、現在の出力が過去から現在までの入力の履歴に依存して決まる論理回路を順序回路 (sequential circuit) と呼ぶ。

#### 組み合わせ回路

出力変数は独立  $\Rightarrow$  複数入力、1 出力の組み合わせ回路が集まったものとして扱える

定義 2.1(論理関数)  $B = 0, 1$  の上で任意の自然数  $n$  に対して定義される関数  $F : B^n \rightarrow B$  を論理関数 という。

### 2.1.2 真理値表と論理式

#### 真理値表 (truth table)

真理値表は全通りの入力組み合わせに対して、出力の値を示す表である。 $n$  入力の場合、 $2^n$  個のエントリーがある。それぞれの出力が 2 通りずつあるので、 $n$  入力論理関数は  $2^{2^n}$  種類ある。

$x_1$	$x_2$	$x_3$	$f$
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

表 2.1 3 入力真理値表の例 ( $f = (x_1 \cdot x_2) \vee (\overline{x_1} \cdot x_3)$ )



## 論理式

論理変数と定数に論理演算を適用して得られる式。以下の論理式は、上の真理値表と対応する。このように論理式は真理関数に対して一意ではない

$$f = \overline{x_1} \cdot \overline{x_2} \cdot x_3 \vee \overline{x_1} \cdot x_2 \cdot x_3 \vee x_1 \cdot x_2 \cdot \overline{x_3} \vee x_1 \cdot x_2 \cdot x_3$$

$$f = x_1 \cdot x_2 \vee \overline{x_1} \cdot x_3 \vee x_2 \cdot x_3$$

$$f = x_1 \cdot x_2 \vee \overline{x_1} \cdot x_3$$

## 2.1.3 完全定義関数と不完全定義関数

すべての入力組み合わせに対して出力が定義されている論理関数を完全定義関数と呼ぶ。ある入力組み合わせに対して出力が定義されていない論理関数を不完全定義関数と呼ぶ。

## 2.2 論理式の標準展開

論理式の一意な表現を得るための式の展開を考える。

## 2.2.1 極小項表現と極大項表現

積項 (product term)

1 つ以上の論理変数またはその否定を論理積だけで結合した論理式を積項と呼ぶ。

$$x_1, x_1 \cdot x_2, \overline{x_1} \cdot x_2$$

積和形 (sum of product)

1 つ以上の積項を論理和で結合した論理式を積和形と呼ぶ。

$$x_1 \cdot x_2 \vee \overline{x_1} \cdot x_3 \vee x_2 \cdot x_3$$

和項、和積形も双対的に定義される。

極小項

入力変数が  $n$  個で  $x_1, x_2, \dots, x_n$  で表される時、すべての  $i$  に対し  $x_i$  または  $\overline{x_i}$  のどちらかを必ず含む積項を極小項と呼ぶ。

極小項表現

重複のない極小項だけを含む積和形を極小項表現とよぶ。

極大項

入力変数が  $n$  個で  $x_1, x_2, \dots, x_n$  で表される時、すべての  $i$  に対し  $x_i$  または  $\overline{x_i}$  のどちらかを必ず含む和項を極大項と呼ぶ。

極大項表現

重複のない極大項だけを含む和積形を極大項表現とよぶ。

一意性

任意の論理関数は極小項表現および極大項表現で一意に表せる。これらは、真理値表を書くか、式展開によって導出できる。

## 2.2.2 ブール展開

任意の論理関数はブール展開を繰り返し適用すると一意な積和標準系が得られる。

定理 2.1  $F(x_1, x_2, \dots, x_n) = \overline{x_1} \cdot F(0, x_2, \dots, x_n) \vee x_1 \cdot F(1, x_2, \dots, x_n)$

定理 2.2  $F(x_1, x_2, \dots, x_n) = (\overline{x_1} \vee F(1, x_2, \dots, x_n)) \cdot x_1 \vee F(0, x_2, \dots, x_n)$

## 2.2.3 リード・マラー標準系

以下の真理値表に従う論理演算子を排他的論理和 (exclusive OR) と呼ぶ。

$x$	$y$	$x \oplus y$
0	0	0
0	1	1
1	0	1
1	1	0

表 2.2 排他的論理和の真理値表

定理 2.3

$$\begin{aligned}
 (x \oplus y) \oplus z &= x \oplus (y \oplus z) \\
 x \oplus y &= y \oplus x \\
 x \cdot (y \oplus z) &= (x \cdot y) \oplus (x \cdot z) \\
 x \oplus 0 &= x \\
 x \oplus x &= 0
 \end{aligned}$$

定理 2.4

$$\begin{aligned}
 x \vee y &= x \oplus y \oplus x \cdot y \\
 \overline{x} &= 1 \oplus x
 \end{aligned}$$

この定理から、いままで  $\vee, \cdot, \overline{\phantom{x}}$  を使って表現してきた論理式は、 $\oplus, \cdot$  だけで表現できる。この2つの演算子のみで表現した標準系がリード・マラー標準系である。

一般形  $f(x_1, x_2, \dots, x_m)$  は

$$\begin{aligned}
 f(x_1, x_2, \dots, x_m) = & \\
 & a_0 \oplus \\
 & a_1 \cdot x_1 \oplus a_2 \cdot x_2 \oplus \dots \oplus a_n \cdot x_n \oplus \\
 & a_{1,2} \cdot x_1 \cdot x_2 \oplus a_{1,3} \cdot x_1 \cdot x_3 \oplus \dots \oplus a_{n-1,n} \cdot x_{n-1} \cdot x_n \oplus \\
 & \vdots
 \end{aligned}$$

と書かれる。

リード＝マラー展開

リード＝マラー標準系は以下のリード＝マラー展開を用いて機械的に導出できる。

$$f(x_1, x_2, \dots, x_n) = x_1 \cdot (f(1, x_2, \dots, x_n) \oplus f(0, x_2, \dots, x_n)) \oplus f(0, x_2, \dots, x_n)$$

## 2.3 BDD 表現

論理関数を 2 分木構造で表したものを BDD(Binary Decision Diagram) と呼ぶ。

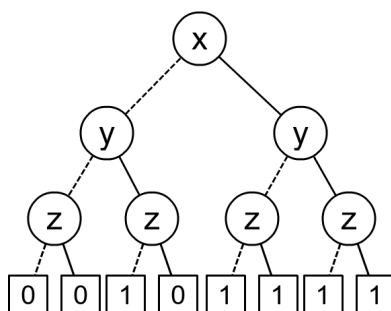


図 2.1 論理式  $x \vee y \cdot \bar{z}$  の BDD。根から始めて、頂点に書かれている変数の値が 1 なら実線、0 なら破線を下向きに辿って行くと値が得られる。

論理式の変数が  $n$  個のとき BDD の頂点数は  $2^n$  になり膨大である。そこで、同じ部分木は一つにまとめることで頂点数を削減したものを ROBDD(Reduced Orderd BDD) と呼ぶ。

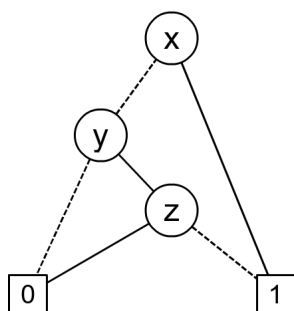


図 2.2 論理式  $x \vee y \cdot \bar{z}$  の ROBDD。

変数の順番を決めれば ROBDD の形は一意に決まる。頂点数の最大値は  $2^n$  だが、現実的なたいていのケースではもっと小さくなる。また ROBDD は論理演算を再帰的に定義しているため、コンピュータで扱いやすい表現と言える。そのうえ、否定をとるのが容易である<sup>\*1</sup>。

<sup>\*1</sup> 極小 (大) 項表現では否定をとるとド・モルガンの公式などを使って多くの計算をしなければならなかった

## 第3章

# 論理関数の性質

### 3.1 ユネイト関数と単調関数

**定義 3.1(包含)** 2つの  $n$  変数関数  $F, G$  と  $0, 1^n$  の任意の要素  $a$  に対して  $F(a) \leq G(a)$  が成立する時  $G$  は  $F$  を包含するといひ  $F(x) \leq G(x)$  と書く。

**定義 3.2(関数の正負)**  $F(x_1, x_2, \dots, x_n)$  において変数  $x_i$  の値を 0 または 1 に置き換えた時  $F(x_1, \dots, x_i, \dots, x_n) \leq F(x_1, \dots, x_i, \dots, x_n)$  が成り立つ時  $F$  は  $X$  に対して正であるという。同様に負も定義できる。

**定理 3.1**  $x_i$  に対して  $F$  が正である必要十分条件は  $F$  が  $\bar{x}$  を含まない和積形の表現を持つことである。

#### ユネイト

論理関数  $F$  がすべての変数に対して正もしくは負の時  $F$  はユネイトであるという。もし、 $F$  がすべての変数に対してせいであるならば  $F$  は正関数であるという。

**定義 3.6** 2つの  $n$  次元 2 値ベクトル  $A = (a_1, a_2, \dots, a_n), B = (b_1, b_2, \dots, b_n)$  が任意の  $i$  に対して  $a_i \leq b_i$  のとき  $A \leq B$  とする。

**定理 3.4**  $n$  変数論理関数  $F$  が正関数であるための必要十分条件は  $A \leq B$  となる任意の  $n$  次元ベクトル  $A, B$  に対して  $F(A) \leq F(B)$  が成立するときである。

### 3.2 双対関数

**定義 3.7(双対関数)**  $n$  変数論理関数に対して  $F_d(x_1, x_2, \dots, x_n) = \overline{F(\bar{x}_1, \bar{x}_2, \dots, \bar{x}_n)}$  を  $F$  の双対関数 (dual function) という。

**定義 3.8(自己 (反) 双対関数)**  $F_d(x) = F(x)$  が成立するとき  $F$  を自己双対関数という。また、 $F_d(x) = \overline{F(x)}$  が成立するとき  $F$  を自己反双対関数という。

**定理 3.8**

$$G(x_1, x_2, \dots, x_n, x_{n+1}) = x_{n+1} \cdot F(x_1, \dots, x_n) \vee \overline{x_{n+1}} F_d(x_1, \dots, x_n)$$

は自己双対関数である。

### 3.3 その他の関数

#### 3.3.1 対称関数

**定義 3.10(対称関数)**  $F(x_1, \dots, x_i, \dots, x_j, \dots, x_n)$  における任意の  $x_i$  と  $x_j$  を入れ替えて得られる関数が元の関数と等しい時  $F$  を対称関数という。

**例 3.17** 任意の 3 変数対称関数は

$$F(x, y, z) = a_0 S_0 \vee a_1 S_1 \vee a_2 S_2 \vee a_3 S_3$$

で表せる。ただし  $a_0 \sim a_3$  は 0 または 1 の定数。

$S_0 = \bar{x} \cdot \bar{y} \cdot \bar{z}$  (入力の中で値が 1 となるものが 0 個の時に出力値が 1 となる関数)

$S_1 = x \cdot \bar{y} \cdot \bar{z} \vee \bar{x} \cdot y \cdot \bar{z} \vee \bar{x} \cdot \bar{y} \cdot z$  (入力の中で値が 1 となるものが 1 個の時に出力値が 1 となる関数)

$S_2 =$  (入力の中で値が 1 となるものが 2 個の時に出力値が 1 となる関数)

$S_3 =$  (入力の中で値が 1 となるものが 3 個の時に出力値が 1 となる関数)

以上から  $n$  変数対称関数は  $2^{n+1}$  通りあることがわかる。

#### 3.3.2 しきい値関数

**定義 3.11(しきい値関数)**

重み (実定数)  $\omega_1, \dots, \omega_n$  としきい値 (実定数)  $T$  が存在し

$$\omega_1 \cdot x_1 + \omega_2 \cdot x_2 + \dots + \omega_n \cdot x_n \geq T \Leftrightarrow F(x_1, \dots, x_n) = 1$$

$$\omega_1 \cdot x_1 + \omega_2 \cdot x_2 + \dots + \omega_n \cdot x_n < T \Leftrightarrow F(x_1, \dots, x_n) = 0$$

と定義される関数をしきい値関数という。

**定義 3.12(多数決関数)**

$$\omega_1 = \omega_2 = \dots = \omega_n = 1$$

$$T = k + 1, n = 2k + 1$$

のとき  $F(x_1, \dots, x_n)$  を多数決関数という。

**定理 3.11** しきい値関数はユネイト関数である。変数  $x_i$  に着目すると

$$\omega_i = 0 \rightarrow x_i \text{ に依存しない} \quad \omega_i > 0 \rightarrow x_i \text{ に対して正} \quad \omega_i < 0 \rightarrow x_i \text{ に対して負}$$

#### 多数決関数

多数決関数  $M(x, y, z) = x \cdot y \vee y \cdot z \vee z \cdot x$  は自己双対関数であり、対称関数であり、単調関数である。(sugoy!)

$M(x, y, 1) = x \vee y, M(x, y, 0) = x \cdot y$  より任意の論理関数は 3 変数の多数決関数と否定で表現できる。

#### 3.3.3 線形関数

**定義 3.13** リード＝マラー標準系に展開した時に論理変数に関して 2 次以上の積項がない論理関数を線形関数と言う。線形関数であることと以下の形に展開できることは同値である。

$$F(x_1, \dots, x_n) = a_0 \oplus a_1 \cdot x_1 \oplus a_2 \cdot x_2 \oplus \dots \oplus a_n \cdot x_n$$

**定理 3.14**  $n$  変数の線形関数は  $2^{n+1}$

定理 3.16 線形関数は自己双対関数または自己反双対関数である。

定義 3.14(ブール微分)

$n$  変数論理関数  $F(x_1, \dots, x_i, \dots, x_n) (1 \leq i \leq n)$  に対して

$$\frac{\partial F}{\partial x_i} := F(x_1, \dots, 0, \dots, x_n) \oplus F(x_1, \dots, 1, \dots, x_n)$$

を  $F$  の  $x_i$  に関するブール微分 という。

論理関数  $F$  の  $x_i$  に関するブール微分が恒等的に 0 のとき  $F$  は  $x_i$  に依存せず、恒等的に 1 のとき  $F$  は  $x_i$  の値が変わると必ず変化する。

定理 3.17 論理関数  $F(x_1, \dots, x_n)$  が線形関数であるための必要十分条件は、すべての変数  $x_i$  に関して

$$\frac{\partial F}{\partial x_i} = a_i (0 \text{ または } 1)$$

が成立することである。

#### 故障検出

$n$  変数論理関数でモデル化できるシステム (例えば電子回路) で故障が起きたとき、いずれかの入力変数が 0 または 1 に固定される縮退故障である場合がある。もし  $x_i$  が 0 に縮退しているかどうか調べたければ、 $F(x_i = 0) \oplus F(x_1, \dots, x_n) = 1$  となる  $x_1, \dots, x_n$  を見つければ良い。これは  $x_i = 1$  かつ  $\frac{\partial F}{\partial x_i} = 1$  を満たすような  $x_1, \dots, x_n$  と同値である。

## 第 4 章

# 論理合成

与えられた論理関数を高性能かつ低コストな、つまり最小のゲート数で実現する論理回路を合成したい。どこまでゲート数が小さくなるかは論理素子の種類による。

### 4.1 AND-OR 2 段形式

- 積和系の論理式に対応した 2 段の論理回路である。
- AND ゲートと OR ゲートで論理を合成することに相当
- NAND ゲートのみを用いた場合にも適用可 (NAND はユニバーサル)

論理合成の課題

- 積項の数を最小化 (2 段目の AND の最小化)
- 各積項に含まれる、変数の数の最小化 (1 段目の OR の最小化)

### 4.2 カルノー図による論理式簡単化

カルノー図 Karnaugh map

カルノー図 (Karnaugh map) は真理値表を図で表現したものである。変数が 5 つを超えると紙には書けなくなる。

		$x_1x_2$			
		00	01	11	10
$x_3x_4$	00	0	1	0	0
	01	0	0	0	0
	11	1	0	1	1
	10	1	0	0	1

図 4.1  $f = \overline{x_1} \cdot x_2 \cdot \overline{x_3} \cdot \overline{x_4} \vee x_1 \cdot x_3 \cdot x_4 \vee \overline{x_2} \cdot x_3$  に対応するカルノー図。囲われている範囲が含意項と対応する。

含意項と主項

定義 4.1 論理関数  $f$  と論理積  $p$  に対して  $p \leq f$  が成り立つならば  $p$  を  $f$  の含意項 (implicant) という。

定義 4.2 論理関数  $f$  の含意項  $p$  に関して  $p$  を構成する変数のどの 1 つを除いても、もはや  $f$  の含意項にならないならば  $p$  を  $f$  の主項 (prime implicant) という。主項はカルノー図において 1 のみが含まれる矩形区間に対応する。

定理 4.1 論理関数  $f$  の最小積和形論理式は  $f$  の主項のみの和で表される。<sup>\*1</sup>

ドントケア (出力が定義されない入力がある場合)

ドントケアがある場合、主項を求めるときは \* を 1 とみなし、必須主項を求めるときは \* を 0 とみなして、通常と同様のカルノー図を書けば良い。

$x_3x_4$	$x_1x_2$			
	00	01	11	10
00	1	1	*	1
01	0	1	*	1
11	0	0	*	*
10	0	1	*	*

図 4.2 ドントケアがある場合のカルノー図。7segLED の左上の点灯と対応する。

$x_3x_4$	$x_1x_2$			
	00	01	11	10
00	1	1	1	1
01	0	1	1	1
11	0	0	1	1
10	0	1	1	1

図 4.3 主項を求めるときのカルノー図。これから、 $f = \overline{x_3x_4} \vee x_2\overline{x_3} \vee x_2\overline{x_4} \vee x_1$  と表せる。

$x_3x_4$	$x_1x_2$			
	00	01	11	10
00	1	1	0	1
01	0	1	0	1
11	0	0	0	0
10	0	1	0	0

図 4.4 必須主項を求めるときのカルノー図。 $f = \overline{x_1x_3x_4} \vee \overline{x_1x_2x_3} \vee \overline{x_1x_2x_4} \vee \overline{x_1x_2x_3} \vee \overline{x_1x_2x_4}$  と表せる。

#### 4.2.1 クワイン・マクラスキー法

クワイン・マクラスキー法はカルノー図と同じ手順を用いて、最小積和形論理式を求めるアルゴリズムである。

手順

(STEP1) すべての主項を生成。

(STEP2) すべての極小項を覆うのに必要な最小の主項の集合を見つける。

(STEP1) の再帰的手続き

手順 1  $f = 1$  またはドントケアとなるすべての極小項の集合を求め、これを 1 次集合とする。肯定変数の数で分割し、手順 2 へ進む

手順  $k(k > 1)$  第  $k-1$  次集合に対して  $xP \vee \overline{x}P = P$  の簡単化。得られた新しい分割を第  $k$  次集合とする。もし  $k-1$  次集合と  $k$  次集合が異なるならば、手順  $k+1$  に進む。そうでないなら  $k$  次集合が関数  $f$  のすべての主項からなる集合である。

(STEP2) の詳細

$f = 1$  となるすべての極小項の集合を  $S$  とする。STEP1 で得た主項との包含関係を記す。そこに、以下の規則を適用し最小被覆集合を求める。<sup>\*2</sup>

規則 1  $S$  に含まれるある極小項を覆う主項がちょうど 1 つだけある場合、その主項は必須主項である。その主項が覆う極小項を  $S$  から除いて STEP2 を継続。

<sup>\*1</sup> 必ずしも主項すべてを使う必要はないことに注意。

<sup>\*2</sup> これらの規則に適用順序の決まりはない。順序によって得られる最小被覆集合が異なることがある。また場合によっては最小被覆集合が求まらない場合もある。そのときは分枝限定法を使うことで決定できる。



規則2 ある主項  $A$  が覆う極小項集合を別の主項  $B$  が覆う極小項集合に完全に含まれていたら、主項  $A$  を削除してSTEP2を継続する。

規則3 ある極小項集合  $C$  を覆うすべての主項が別の極小項  $D$  を覆うとき、 $S$  から  $D$  を削除。

STEP1 具体例

$x_1$	$x_2$	$x_3$	$f$
0	0	0	*
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	*
1	1	0	1
1	1	1	0

手順1 以下の極小項の集合が1次集合になる。肯定変数の個数で昇順に並べている。

$$\{\overline{x_1x_2x_3}, \overline{x_1x_2}x_3, \overline{x_1}x_2x_3, x_1\overline{x_2}x_3, x_1x_2\overline{x_3}\}$$

手順2 肯定変数の個数が1つだけ異なる2つについて、 $xP \vee \overline{x}P = P$  の簡単化ができないか試行する。

$$\{\overline{x_1x_2}, \overline{x_1}x_3, \overline{x_2}x_3, x_1x_2\overline{x_3}\}$$

手順3 もうこれ以上簡単化できない。得られた集合が主項の集合である。

## 第 5 章

# 順序回路

順序回路 (sequential circuit) は、入力履歴を内部状態として記憶する論理回路である。

### 5.1 オートマトン順序回路

オートマトン (automaton) はギリシャ語で自動機械の意である。計算機科学の文脈では以下のような性質のうちいくつかを満たすシステムの抽象モデルを指す。

1. 外部からの情報を保持する。
2. 内部に状態を保持する。
3. 外部へ情報を出力する。
4. 外部からの入力と現在の内部状態で次の内部状態が決まる。
5. 外部からの入力と現在の内部状態で外部への出力が決まる。

一般的には 1,2,4 の性質を満たすものをオートマトンと言う。有限状態<sup>\*1</sup>のオートマトンは

$$M = (K, \Sigma, \delta, q_0, F)$$

$K$  : 状態集合

$\Sigma$  : 状態集合

$\delta$  : 状態集合

$q_0$  : 初期状態

$F$  : 最終状態集合

と表される。

出力付きのオートマトン

我々の興味は、一般のオートマトンに加えて 3,5 の性質を満たす出力付きのオートマトンにある。

---

<sup>\*1</sup> 有限種類の内部状態しかとらないことを意味する。

## 5.2 順序回路の実現

### 5.2.1 形式定義

ミーリー型 (Mealy-type)

以下の5項の組で定義される。

$$M = (X, Q, Z, \delta, \omega)$$

$X$  : 入力の集合

$Q$  : 状態の集合

$Z$  : 出力の集合

$\delta : X \times Q \rightarrow Q$  なる状態遷移関数

$\omega : X \times Q \rightarrow Z$  なる出力関数

ムーア (Moore-type)

以下の5項の組で定義される。ミーリー型との差異は  $\omega$  のみである。

$$M = (X, Q, Z, \delta, \omega)$$

$X$  : 入力の集合

$Q$  : 状態の集合

$Z$  : 出力の集合

$\delta : X \times Q \rightarrow Q$  なる状態遷移関数

$\omega : Q \rightarrow Z$  なる出力関数

## 5.3 フリップフロップ

記憶素子 (フリップフロップ) は2つの内部状態を持つ記憶素子である。与えられた論理関数を実現する順序回路を作る場合、採用する記憶素子の動作に依存してその設計は変わってくる。

(ラッチ (非同期式フリップフロップ))

ラッチ (Latch) は2入力 ( $S, R$ )、2出力 ( $Q, \bar{Q}$ ) 型のフリップフロップである。以下の性質を満たす

- $S = R = 0$ :  $Q, \bar{Q}$  は値を維持。状態を保つ。
- $S = 1, R = 0$ :  $Q = 1, \bar{Q} = 0$ 。状態を Set する。
- $S = 0, R = 1$ :  $Q = 0, \bar{Q} = 1$ 。状態 Reset する。
- $S = R = 1$ : 禁止入力。

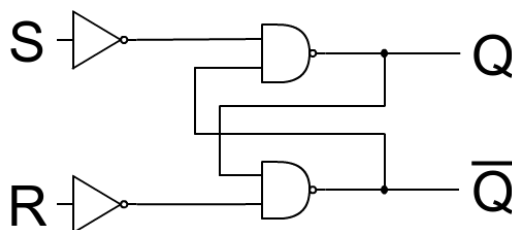


図 5.1 ラッチを実現する回路

## フリップフロップ (同期式フリップフロップ)

フリップフロップはクロック信号に同期して状態が変化する記憶素子である。状態遷移はクロック毎に発生する。

## (1)SR フリップフロップ

入力信号が安定しているときに十分幅の短いクロックが入力されるという仮定のもとで、以下のような入力駆動条件を持つフリップフロップ回路である。

$Q_t$	$Q_{t+1}$	$S$	$R$
0	0	0	*
0	1	1	0
1	0	0	1
1	1	*	0

表 5.1 SR フリップフロップの入力駆動条件

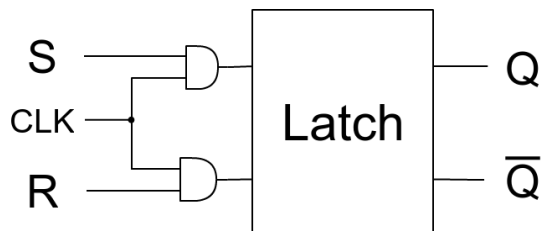


図 5.2 SR フリップフロップ実現する回路

## (2)D フリップフロップ

D フリップフロップはクロック入力と D 入力からなるフリップフロップである。動作として、D の入力が 1 なら次の状態は 1 であり、D の入力が 0 ならば次の入力が 0 になる。

$Q_t$	$Q_{t+1}$	$D$
0	0	0
0	1	1
1	0	0
1	1	1

表 5.2 D フリップフロップの入力駆動条件

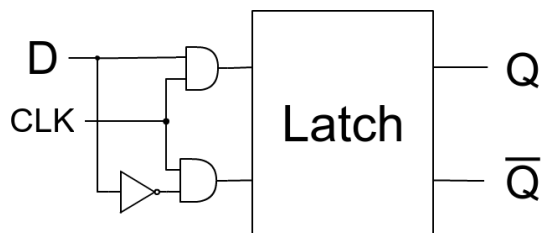


図 5.3 D フリップフロップ実現する回路

## (3)T フリップフロップ

そういうのがある

## (4)JK フリップフロップ

JK フリップフロップはクロック入力と J 入力と K 入力からなるフリップフロップである。入力駆動条件は SR フリップフロップとほぼ同じだが、 $J = K = 1$  となる入力が新たに許可されている。 $J = K = 1$  の場合状態を反転している。

$Q_t$	$Q_{t+1}$	$J$	$K$
0	0	0	*
0	1	1	*
1	0	*	1
1	1	*	0

表 5.3 SR フリップフロップの入力駆動条件

## マスタースレーブ方式フリップフロップ

今までの回路では「入力が安定時に十分幅の短いクロックが入る」という仮定がされていた。これはフリップフロップの出力が安定する前に入力が変化すると誤動作する可能性があるというタイミング問題のためである。マスタースレーブ方式フリップフロップはその仮定が成立しない時でもうまく動作する。マスタースレーブ方式のフリップフロップは、マスターフリップフロップとスレーブフリップフロップの2つで構成される。スレーブのクロック入力はマスターのクロック入力の反転を与える。

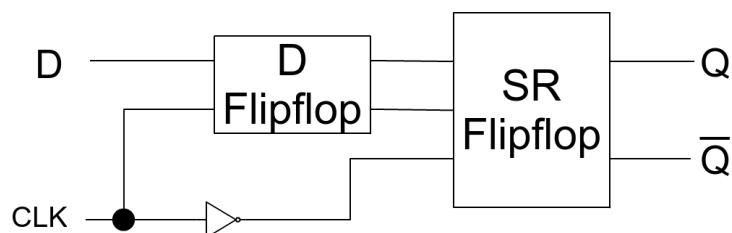


図 5.4 マスタースレーブ方式の D フリップフロップ。D フリップフロップをマスターとし、SR フリップフロップをスレーブとしている。

## エッジトリガ方式フリップフロップ

エッジトリガ方式フリップフロップ (Edge-Trigger) はクロック信号の立ち上がり時点の入力のみで次の状態を決定するフリップフロップである。エッジトリガ方式フリップフロップを実現する回路図は複雑である。

## 5.3.1 順序回路の合成

与えられた動作仕様から順序回路を導出することを考えたい。以下のようなプロセスで導出することができる。

- (1) 動作仕様を状態遷移図などの形式的記述で与える。
- (2) 状態遷移図を簡単化する。
- (3) 使用するフリップフロップの種類と数を決定する。
- (4) 入力駆動条件を用いて状態遷移関数と出力関数を導出。

## 5.4 順序回路の等価性

内部状態が観測できない順序回路が与えられたとき、ある 2 つの状態が等価であるかどうか調べたい。内部状態が異なっても、入出力が同じならば等価と呼んでも良いかもしれない。ここで等価性の議論のために、どういうときに等価と呼ぶかを定義する。

### 状態の等価性

定義 5.12 つの状態  $q_i$  と  $q_j$  に同じ入力系列  $X$  を加えた時に、出力系列が異なれば  $X$  は  $q_i$  を  $q_j$  を区別するという。定義 5.22 つの状態  $q_i$  と  $q_j$  を区別する入力系列が存在しない時、 $q_i$  と  $q_j$  は等価であるという。

### 状態の $s$ 次等価 ( $S \in \mathbb{N}$ )

定義 5.32 つの状態  $q_i$  と  $q_j$  を区別する長さ  $S$  以下の入力系列が存在しないとき  $q_i$  と  $q_j$  は  $S$  次等価であるという。

定理 5.1  $S$  次分割の同一ブロックに属する 2 つの状態  $q_i$   $q_j$  について遷移先が異なるブロックに属する入力が存在しないなら、その時に限り  $q_i$  と  $q_j$  は  $(S + 1)$  次等価である。

### 等価な状態の求め方

(step0) 1 次分割を行う。つまり 1 次等価な状態をグループに分割。step1 にすすむ。

(step $k$ ) 定理 5.1 を用い  $k$  次分割から  $k + 1$  次分割を導出する。もし  $k$  次分割と  $k + 1$  次分割が等しいならば終了する。

そうでないならば step( $k + 1$ ) に進む。

状態	0	1
$q_0$	$q_3/0$	$q_1/0$
$q_1$	$q_4/0$	$q_2/1$
$q_2$	$q_4/0$	$q_3/1$
$q_3$	$q_0/0$	$q_4/0$
$q_4$	$q_2/0$	$q_3/1$

1 次分割				2 次分割				3 次分割			
状態		入力 0	入力 1	状態		入力 0	入力 1	状態		入力 0	入力 1
$G(1)A$	$q_0$	$G(1)A$	$G(1)B$	$G(2)A$	$q_0$	$G(2)A$	$G(2)B$	$G(3)A$	$q_0$	$G(3)B$	$G(1)B$
	$q_3$	$G(1)A$	$G(1)B$		$q_3$	$G(2)A$	$G(2)C$		$q_3$	$G(3)A$	$G(1)D$
$G(1)B$	$q_1$	$G(1)B$	$G(1)B$	$G(2)B$	$q_1$	$G(2)C$	$G(2)C$	$G(3)C$	$q_1$	$G(3)D$	$G(1)D$
	$q_2$	$G(1)B$	$G(1)A$		$q_2$	$G(2)C$	$G(2)A$		$q_2$	$G(3)D$	$G(1)B$
	$q_4$	$G(1)B$	$G(1)A$		$q_4$	$G(2)C$	$G(2)A$		$q_4$	$G(3)D$	$G(1)B$

## 第 6 章

# 演算回路

この章では、数値表現の方法や基本演算のアルゴリズム、そのハードウェア構成を学ぶ。

## 6.1 データ表現

### 6.1.1 データの符号化

数値データ

数値データは 2 進数で表現する。

非数値データ

ASCII(American Standard Code for Information Interchange) → ISO8859 等号  
などいろいろなルールにしたがって数値化する。

### 6.1.2 負数の表現 (整数の場合)

$n$  bit で整数を表現する時、符号をどのように表すかはいくつかの方法がある。

#### (1) 符号付き絶対値表現

符号付き絶対値表現では最上位 bit を符号を表現するために使い、残りの bit でその値の絶対値を表現するような方法である。 $-2^{n-1} \sim 2^{n-1}$  を表現する。この方法は人間にはわかりやすいが、演算処理をするときに、符号を比べたり、大小関係を比べたりする必要があり、ハードウェアに取っては少し面倒である。

#### (2-1) 1 の補数表現

1 の補数表現では、正の数は通常通り表現し、負の数  $-N$  は  $(2^n - 1) - N$  として表記する方法である。これは絶対値のビット反転と同じである。

#### (2-2) 2 の補数表現

2 の補数表現で、正の数は通常通り表現し、負の数  $-N$  は  $2^n - N$  として表記する方法である。これは絶対値のビット反転 + 1 と同じである。<sup>\*1</sup>2 の補数表現であれば、正負を考慮した加減算の結果が、表記上の加減算と一致する。というのも、表現と実際に表した値は  $\text{mod}(2^n)$  で等しくなっているからである。

---

<sup>\*1</sup> 1 の補数表現における「1」は全ての桁が 1 であるビットとの差をとることを表しており、2 の補数表現における 2 は「2」進数表記における補数であることを表している。これは一般の進数に対して拡張可能であるが、2 の補数と言った時に、2 進数における 2 の補数なのか、3 進数における 2 の補数なのか区別がつかず、とてもつらい。英語の場合 two's complement と twos' complement という風に区別をつけることができる。

## (3) バイアス表現

バイアス表現ではある定数  $B$  を用いて、表現したい値  $N(-B \leq N)$  を  $N + B$  として表記する表現である。

表現	1 の補数表現での値	2 の補数表現での値
000	0	0
001	1	1
010	2	2
011	3	3
100	-3	-4
101	-2	-3
110	-1	-2
111	0	-1



## 参考文献

[1] ... ..