情報工学実験１
実験３
汎用ロジックICによる組み合わせ回路の実現

035764C 若津 大悟

グループH
実験実施日：平成１６年6月１日
提出〆切り日：平成１６年6月８日
共同実験者名
035762G：吉永　安磨
035763E：饒平名隆一

1　実験目的

本実験では、簡単な組み合わせ回路の設計および実現を行うことによって、カルノー図などを用いた論理関数の簡単化に慣れるとともに、実際のコンピュータに使用されている演算器の設計法について習得する。

1.1　関連授業

- ディジタル回路

2　実験

(1) 半加算器、全加算器、２ビット加算器の真理値表を書き、それぞれの論理関数を求めよ。

1. 半加算器

入力		出力	
A	B	c	s
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

表 1: 半加算器の真理値表

◯ 真理値表より論理関数を求める。(加法標準形)

$$
\begin{aligned}
c &= A \cdot B \\
s &= \overline{A} \cdot B + A \cdot \overline{B} \\
(&= A \otimes B)
\end{aligned}
$$

2. 全加算器

入力			出力	
A	B	C’	c	s
0	0	0	0	0
0	1	0	0	1
1	0	0	0	1
1	1	0	1	0
0	0	1	0	1
0	1	1	1	0
1	0	1	1	0
1	1	1	1	1

表 2: 全加算器の真理値表

◯ 真理値表より論理関数を求める。(加法標準形)

$$
\begin{aligned}
c &= \overline{A} \cdot B \cdot C' + A \cdot \overline{B} \cdot C' + A \cdot B \cdot \overline{C'} + A \cdot B \cdot C \\
s &= \overline{A} \cdot \overline{B} \cdot C' + \overline{A} \cdot B \cdot \overline{C'} + A \cdot \overline{B} \cdot \overline{C'} + A \cdot B \cdot C
\end{aligned}
$$

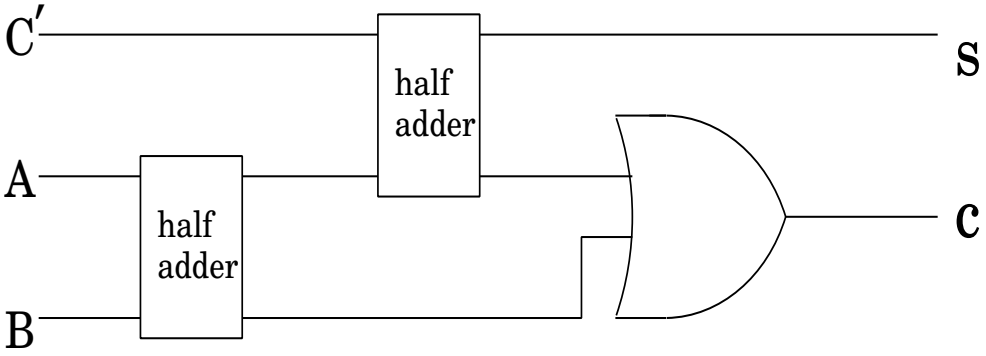

図 1: 全加算器の構成図

3. ２ビット加算器

入力				出力		
a_1	a_0	b_1	b_0	c	s_1	s_0
0	0	0	0	0	0	0
0	0	0	1	0	0	1
0	0	1	0	0	0	1
0	0	1	1	0	1	0
0	1	0	0	0	0	1
0	1	0	1	0	1	0
0	1	1	0	0	1	0
0	1	1	1	0	1	1
1	0	0	0	0	0	1
1	0	0	1	0	1	0
1	0	1	0	0	1	0
1	0	1	1	0	1	1
1	1	0	0	0	1	0
1	1	0	1	0	1	1
1	1	1	0	0	1	1
1	1	1	1	1	1	0

表 3: ２ビット加算器の真理値表

◯ 真理値表により、論理関数を求める。(加法標準形)

$$
\begin{aligned}
c &= \overline{a_1}\cdot a_0\cdot b_1\cdot b_0 + a_1\cdot\overline{a_0}\cdot b_1\cdot\overline{b_0} + a_1\cdot\overline{a_0}\cdot b_1\cdot b_0 \\
&+ a_1\cdot a_0\cdot\overline{b_1}\cdot b_0 + a_1\cdot a_0\cdot b_1\cdot\overline{b_0} + a_1\cdot a_0\cdot b_1\cdot b_0 \\
s_1 &= \overline{a_1}\cdot\overline{a_0}\cdot b_1\cdot\overline{b_0} + \overline{a_1}\cdot\overline{a_0}\cdot b_1\cdot b_0 + \overline{a_1}\cdot a_0\cdot\overline{b_1}\cdot b_0 + \overline{a_1}\cdot a_0\cdot b_1\cdot\overline{b_0} \\
&+ a_1\cdot\overline{a_0}\cdot\overline{b_1}\cdot\overline{b_0} + a_1\cdot\overline{a_0}\cdot\overline{b_1}\cdot b_0 + a_1\cdot a_0\cdot\overline{b_1}\cdot\overline{b_0} + a_1\cdot a_0\cdot b_1\cdot b_0 \\
s_0 &= \overline{a_1}\cdot\overline{a_0}\cdot\overline{b_1}\cdot b_0 + \overline{a_1}\cdot\overline{a_0}\cdot b_1\cdot b_0 + \overline{a_1}\cdot a_0\cdot\overline{b_1}\cdot\overline{b_0} + \overline{a_1}\cdot a_0\cdot b_1\cdot\overline{b_0} \\
&+ a_1\cdot\overline{a_0}\cdot\overline{b_1}\cdot b_0 + a_1\cdot\overline{a_0}\cdot b_1\cdot b_0 + a_1\cdot a_0\cdot\overline{b_1}\cdot\overline{b_0} + a_1\cdot a_0\cdot b_1\cdot\overline{b_0}
\end{aligned}
$$

(2) 実験 (1) で作成した各真理値表をもとにカルノー図を描き、簡単化された論理関数を求めよ

1. 半加算器

B \ A	0	1
0	0	0
1	0	1

図 2: 半加算器 (出力 c)

B \ A	0	1
0	0	1
1	1	0

図 3: 半加算器 (出力 s)

◯ 簡単化された論理関数 (半加算器)

$$
\begin{aligned}
c &= A \cdot B \\
s &= \overline{A} \cdot B + A \cdot \overline{B}
\end{aligned}
$$

2. 全加算器

B \ AB	00	01	11	10
0	0	0	1	0
1	0	1	1	1

図 4: 全加算器 (出力 c)

B \ AB	00	01	11	10
0	0	1	0	1
1	1	0	1	0

図 5: 全加算器 (出力 s)

◯ 簡単化された論理関数 (全加算器)

$$
\begin{aligned}
c &= A \cdot B + B \cdot C' + A \cdot C' \\
s &= \overline{A} \cdot B \cdot \overline{C'} + A \cdot \overline{B} \cdot \overline{C'} + \overline{A} \cdot \overline{B} \cdot C' + A \cdot B \cdot C'
\end{aligned}
$$

3. 2 ビット加算器

b_1b_2 \ a_1a_2	00	01	11	10
00	0	0	1	1
01	0	1	0	1
11	1	0	1	0
10	1	1	0	0

図 6: 2 ビット加算器 (出力 s_1)

b_1b_2 \ a_1a_2	00	01	11	10
00	0	1	1	0
01	1	0	0	1
11	1	0	0	1
10	0	1	1	0

図 7: 2 ビット加算器 (出力 s_0)

b_1b_2 \ a_1a_2	00	01	11	10
00	0	0	0	0
01	0	1	1	0
11	0	1	1	1
10	0	0	1	0

図 8: 2 ビット加算器 (出力 c)

○ 簡単化された論理関数 (2 ビット回路)

$$
\begin{aligned}
c &= a_1 \cdot a_0 + a_1 \cdot b_1 \cdot b_0 + a_1 \cdot a_0 \cdot b_1 \\
s_0 &= a_0 \cdot \overline{b_1} + \cdot \overline{a_0} \cdot b_0 \\
s_1 &= a_1 \cdot \overline{b_1} \cdot \overline{b_0} + a_1 \cdot \overline{a_0} \cdot \overline{b_1} + \overline{a_1} \cdot a_0 \cdot b_1 \\
&+ \overline{a_1} \cdot b_1 \cdot \overline{b_0} + \overline{a_1} \cdot a_0 \cdot \overline{b_1} \cdot b_0 + a_1 \cdot a_0 \cdot b_1 \cdot b - 0
\end{aligned}
$$

(3) 実験 (1) および実験 (2) で得られた論理関数を比較し、実験 (2) で得られた論理関数が簡単化されていることを確認せよ。

カルノー図をで考えることにより、半加算器は変わらなかったが、全加算器の出力 C や、２ビット加算器の出力 s_0 などは、簡単化されることが確認できた。
非常に便利な方法だと感じた。

(4) 実験 (2) で得られた論理関数から、半加算器、全加算器、２ビット加算器の回路図を書け。

○ 半加算器の回路図

カルノー図によって得られた論理関数から回路図を作成し、実験しやすいようにそれを変形してみた。
結果

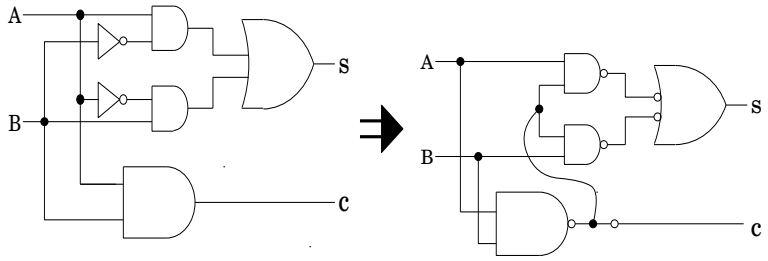

図 9: 半加算器の回路図を変形

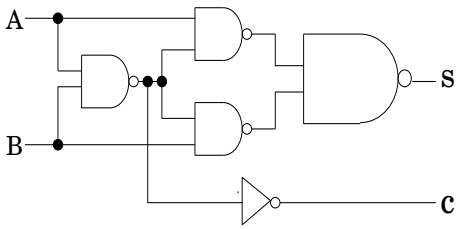

図 10: NOT と NAND で表した半加算器の回路図

◯ 全加算器の回路図

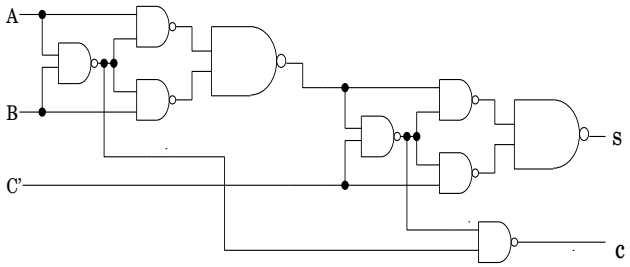

図 11: 全加算器の回路図

◯ 2 ビット回路図の出力 c と出力 s_1 の回路図

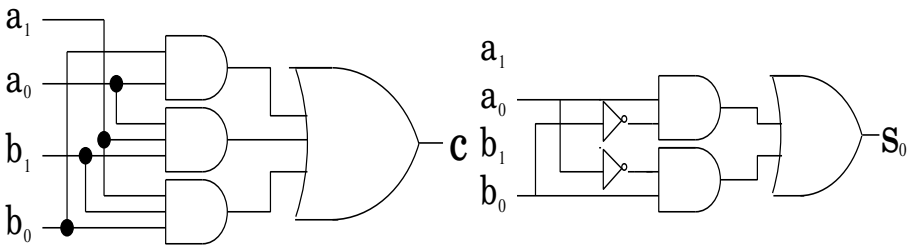

図 12: 2 ビット加算器の回路図

(5) 実験 (4) で描いた回路図をもとにして、ブレッドボード上に半加算器を実現し、その動作を確認せよ

図 10. 半加算器の回路図より、4011B 規格の NAND ゲート IC と、4069UB 規格の NOT ゲート IC を用いてブレッドボード上に実現した。真理値表と同じ動作が確認できた。

(6) 実験 (4) で描いた回路図をもとにして、ブレッドボード上に全加算器を実現し、その動作を確認せよ。

図 11. 全加算器の回路図より、4011B 規格の NAND ゲート IC 2つと、4069UB 規格の NOT ゲート IC を用いてブレッドボード上に実現した。真理値表と同じ動作が確認できた。

$$
\begin{aligned}
c &= A \cdot B + B \cdot C' + A \cdot C' \\
s &= \overline{A} \cdot B \cdot \overline{C'} + A \cdot \overline{B} \cdot \overline{C'} + \overline{A} \cdot \overline{B} \cdot C' + A \cdot B \cdot C'
\end{aligned}
$$

3　実用的な組み合わせ回路

加算器以外の実用的な組み合わせ回路を 3 つ挙げ、それらの真理値表と回路図を示せ。また、それらの動作や特徴について述べよ。

1. エンコーダ

   エンコーダとは、データを一定の規則に基づいて符号化し、データの圧縮や暗号化などを行う。例えば、10 進数を 2 進化 10 進符号に変換したり、8 進数を 2 進数に変換したりする。

	１０進入力										２進出力			
	x_0	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	y_3	y_2	y_1	y_0
0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	1	0	0	0	0	0	0	0	0	0	0	0	1
2	0	0	1	0	0	0	0	0	0	0	0	0	1	0
3	0	0	0	1	0	0	0	0	0	0	0	0	1	1
4	0	0	0	0	1	0	0	0	0	0	0	1	0	0
5	0	0	0	0	0	1	0	0	0	0	0	1	0	1
6	0	0	0	0	0	0	1	0	0	0	0	1	1	0
7	0	0	0	0	0	0	0	1	0	0	0	1	1	1
8	0	0	0	0	0	0	0	0	1	0	1	0	0	0
9	0	0	0	0	0	0	0	0	0	1	1	0	0	1

表 4: 10 進数を 2 進数に変換するエンコーダ

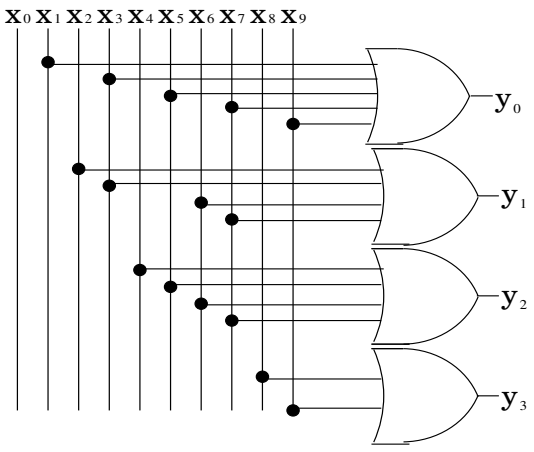

図 13: 10 進数を 2 進数に変換するエンコーダの回路図

2. デコーダ

デコーダとは、一定の規則に基づいて符号化されたデータを復号し、もとのデータを取り出し、圧縮されたデータの復元や、暗号の解読などを行なう。2 進数を 8 進数や 10 進数に変換したり、2 進化 10 進符号を 10 進数に変換したりする。

	2進出力				10進入力									
	x_3	x_2	x_1	x_0	y_0	y_1	y_2	y_3	y_4	y_5	y_6	y_7	y_8	y_9
0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
1	0	0	0	1	0	1	0	0	0	0	0	0	0	0
2	0	0	1	0	0	0	1	0	0	0	0	0	0	0
3	0	0	1	1	0	0	0	1	0	0	0	0	0	0
4	0	1	0	0	0	0	0	0	1	0	0	0	0	0
5	0	1	0	1	0	0	0	0	0	1	0	0	0	0
6	0	1	1	0	0	0	0	0	0	0	1	0	0	0
7	0	1	1	1	0	0	0	0	0	0	0	1	0	0
8	1	0	0	0	0	0	0	0	0	0	0	0	1	0
9	1	0	0	1	0	0	0	0	0	0	0	0	0	1

表 5: 2 進数を 10 進数に変換するデコーダ

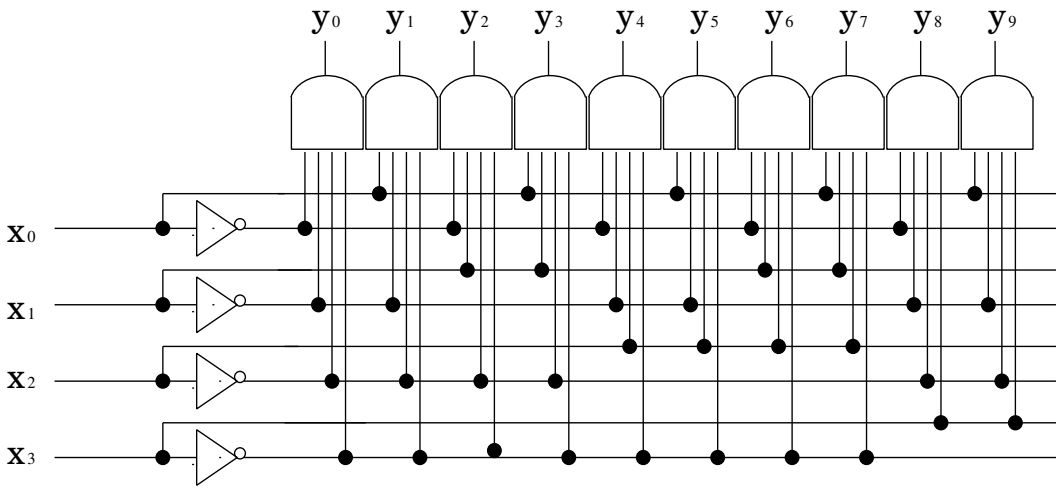

図 14: 2 進数を 10 進数に変換するエンコーダの回路図

3. 7セグメント数字表示回路

1ビットの2進符号の入力に対して、図〜に示す7セグメントの10進数0〜9を表示させる論理回路を考える。真理地表より、10進数で10〜14までに対応する2進符号入力については、E(error)を表示し、10進数で15すなわち2進符号で1111を入力すると、全てのセグメントが点灯して、ランプのテストができるようにする。

<次ページ 真理値表>

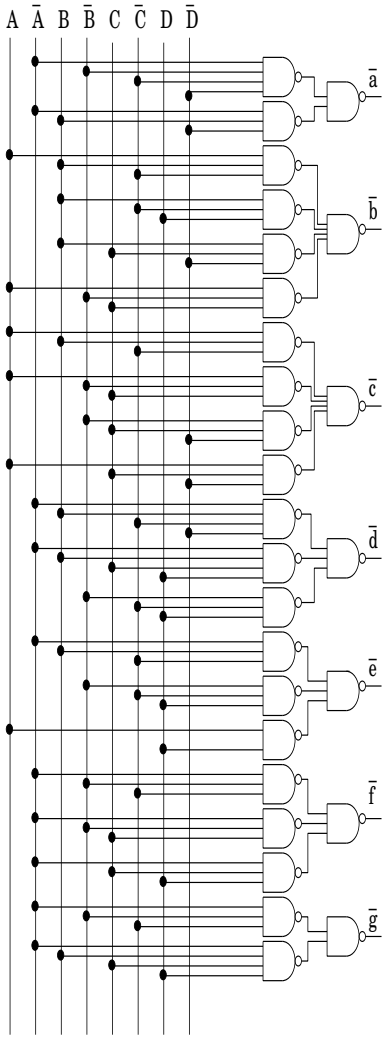

図 15: 7セグメント数字表示回路

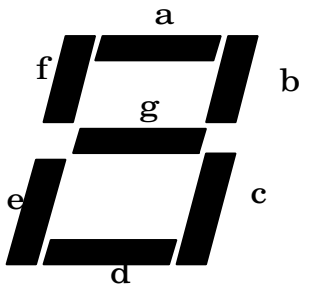

図 16: 7セグメント数字表示

	入力				出力						
	A	B	C	D	a	b	c	d	e	f	g
0	0	0	0	0	1	1	1	1	1	1	0
1	0	0	0	1	0	1	1	0	0	0	0
2	0	0	1	0	1	1	0	1	1	0	1
3	0	0	1	1	1	1	1	1	0	0	1
4	0	1	0	0	0	1	1	0	0	1	1
5	0	1	0	1	1	0	1	1	0	1	1
6	0	1	1	0	0	0	1	1	1	0	1
7	0	1	1	1	1	1	1	0	0	1	0
8	1	0	0	0	1	1	1	1	1	1	1
9	1	0	0	1	1	1	1	0	0	1	1
E	1	0	1	0	1	0	0	1	1	1	1
E	1	0	1	1	1	0	0	1	1	1	1
E	1	1	0	0	1	0	0	1	1	1	1
E	1	1	0	1	1	0	0	1	1	1	1
E	1	1	1	0	1	0	0	1	1	1	1
8	1	1	1	1	1	1	1	1	1	1	1

表 6: 7 セグメント数字表示回路の真理値表

4　キャリールックアヘッド方式加算器

加算を高速に行なえる加算器の一つにキャリールックアヘッド方式の加算器がある。キャリールックアヘッド方式の加算器とは、どのような加算器か調査し説明せよ。

◯ キャリールックアヘッド方式の加算器とは、桁上げ先見加算器のこと

桁上がりを先行して計算することにより、計算処理全体の動作が高速な加算器になり、桁上がりを先に計算する回路を別に持っている。通常、乗算などの、加算を繰り返して計算するなどの高速な計算を必要とするときに使用される。

5　同期式と非同期式

順序回路は、同期式と非同期式に大別される。両者の違いを調査し報告せよ。

◯ 順序回路とは

まず、順序回路とは、出力が現時点での入力だけでは決まらず、過去の入力にも依存する論理回路のことを言う。過去の入力を記憶していると考えると、順序回路は記憶を持つ回路であり、出力が現時点での入力と記憶に依存する回路であるとも言える。組み合わせ回路の出力が現時点での入力のみによって決まるのと対照的である。

◯ 同期式と非同期式、クロック入力

順序回路は、その内部に記憶回路を含んでおり、出力の記憶は入力が変化すると共に変化する。つまり、出力や記憶が現時点での入力と出力によって決まることにならない。よって、現時点での入力と記憶を取り入れて、出力や記憶は変化させるが、それらの変化後の値は取り込まないようなメカニズムが必要となる。そのために、通常の入力の他に、記憶回路に入力を取り込むことを指示するための入力を、別に記憶回路に加えることにする。

この入力は、クロック入力とよばれ、一定の時間間隔毎に信号変化を送る動作をする。また、クロックを一定時間ごとに加えることにしたときの周波数をクロック周波数と呼ぶ。
このように、クロックを使用する順序回路を同期式順序回路と呼び、そうでないものを非同期式順序回路と呼ぶ。

6　D フリップフロップおよび (同期式) カウンタ

D フリップフロップおよび (同期式) カウンタとはどのような回路か調査し報告せよ。

フリップフロップは、クロック入力 CK が 1 になったとき、入力 D の値と同一の出力が Q に得られるもので、入力 D の情報をクロックパルスが入力されるまで遅らせる働きをする。
データフリップフロップ、あらいは遅延フリップフロップともよばれる。

D	Q^{n+1}
0	0
1	1

表 7: D フリップフロップの特性

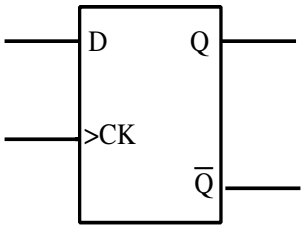

図 17: D フリップフロップ

◯ カウンタとは

カウンタとは、それに入力されたクロックパルスの数を数えて、その数を２進数として出力する回路である。

このとき、n 個のパルスを $0, 1, 2 \cdots, n-1, 0, 1, 2, \cdots$ と順に繰り返し数えるカウンタを n 進カウンタという。

7　考察

本実験について考察せよ。

論理関数を考える時、式を展開して簡単化することは、論理ゲートを効率良く実現する上で、非常に重要である。このとき、プール式をそのまま展開しようとすると、非常に時間がかかってしまった。そこで、カルノー図を描いて、カルノー図の’１’の部分に注目し、’１’が 2^i 個隣接しているところを一まとまりにして考え、それを参照することにより、式を効率良く簡単化することができる。

しかし、カルノー図によって簡単化したこの式で、ブレッドボード上にゲートを実現することを考えると、簡単化されたこの論理式は、AND ゲートと、OR ゲート、NOT ゲートでしか表していないので、IC を使ってゲートを実現するということを考慮すると、AND ゲートや OR ゲート (トランジスタ数６個 (N 型 MOS トランジスタ３つ、P 型 MOS トランジスタ３つ)) より、トランジスタ数の少ない、NAND ゲートや、NOR ゲート (トランジスタ数４個 (N 型 MOS トランジスタ２つ、P 型 MOS トランジスタ２つ)) を用いた方が、より簡単な、効率の良い回路が実現できる。

よって、論理関数を考える時は、まず、その真理値表を書き、それを参照してカルノー図を描く、カルノー図で、式を簡単化する、そして最後にどうすれば少ないトランジスタ数でゲートを実現できるかを考え、最後に効率良い論理式になるように、プール式を展開していく、という方法で考えていけば良い。

8　参考文献・URL

- ディジタル電子回路： 工学博士 藤井信生 著、昭晃堂
- 論理回路1:http://laputa.cs.shinshu-u.ac.jp/~yizawa/logic_textbook.html

9　実験に使用した器具

- 直流電源
- IC :NAND TC4081BP
- :NOT TC4069UBP
- ブレッドボード・導線...