情報科学実験 1
1. 算術演算回路

実験グループ: A 班
報告者: J2200071 齊藤 隆斗
共同実験者: J2200038 小淵 萌, J2200158 矢島琴恵

実験実施日: 2024 年 5 月 23 日
レポート提出日: 2024 年 5 月 30 日
提出期限: 2024 年 5 月 30 日

1　実験の目的

ディジタル計算機においては、基本的な演算処理として、数値やデータ間での比較、加算、減算、乗算、そして除算といった 2 項演算が不可欠である. これらの演算は、計算機の根幹をなす基本的な機能であり、ディジタル計算機が複雑な計算を行う上で欠かせないプロセスを形成している. 本実験では、これらの基本的な 2 項演算が、どのようにして論理回路の組み合わせによって実現されるのかについて探求する. この実験を通じて、ディジタル計算機の基本的な動作原理を理解し、論理回路の設計や構築に関する知識を深めることが目的である.

2　原理

2.1　比較器

2 つの数の大小を判別する回路を比較器という.1bit の正の 2 進数 A, B に対する大小関係を調べた結果は表 1 のようになる.

表 1　2 数 A,B の比較

A	B	A > B	A < B	A = B
0	0	0	0	1
0	1	0	1	0
1	0	1	0	0
1	1	0	0	1

この結果から、$A > B, A < B, A = B$ の 3 つの状態を表す論理関数をそれぞれ f_0, f_1, f_2 とおくと、

$$f_0 = A \cdot \overline{B} \tag{1}$$
$$f_1 = \overline{A} \cdot B \tag{2}$$
$$f_2 = \overline{A} \cdot \overline{B} + A \cdot B = \overline{f_0} \cdot \overline{f_1} \tag{3}$$

ということがわかる.

2.2　加算

2.2.1　半加算器

1 桁の 2 進数の数 A,B を加え、和 S_0 と上位への桁上げ信号 C_0 とを出力する回路を半加算器という. その真理値表を表 2 に示す.

このことから、次の論理式が求まる.

$$S_0 = A \cdot \overline{B} + \overline{A} \cdot B = A \oplus B \tag{4}$$
$$C_0 = A \cdot B \tag{5}$$

表 2　半加算器の真理値表

A	B	S_0	C_0
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

2.2.2　全加算器

被加数 A と加数 B および下位からの桁上げ C_{n-1} の 3 つの入力に対して、和 S_n と上位への桁上げ C_n を出力する回路を全加算器という. その真理値表を表 3 に示す.

表 3　全加算器の真理値表

A	B	C_{n-1}	S_n	C_n
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

この表より、カルノー図を描くと表 4, 5 のようになるので、次の論理式が求まる.

$$S_n = \overline{A} \cdot \overline{B} \cdot C_{n-1} + \overline{A} \cdot B \cdot \overline{C}_{n-1} + A \cdot \overline{B} \cdot \overline{C}_{n-1} + A \cdot B \cdot C_{n-1} \tag{6}$$
$$C_n = A \cdot B + B \cdot C_{n-1} + A \cdot C_{n-1} \tag{7}$$

排他的論理和を用いて変形すると

$$S_n = A \oplus B \oplus C \tag{8}$$
$$C_n = A \cdot B + (A \oplus B) \cdot C_{n-1} \tag{9}$$

となる.

さらにこの式を半加算器の論理式を用いて変形すると、

$$S_n = S_0 \oplus C_{n-1} \tag{10}$$
$$C_n = C_0 + S_0 \cdot C_{n-1} \tag{11}$$

となる.

表 4　全加算器のカルノー図 (S_n)

	0	1
0 0	0	1
0 1	1	0
1 0	0	1
1 1	1	1

表 5　全加算器のカルノー図 (C_n)

	0	1
0 0	0	0
0 1	0	1
1 0	1	1
1 1	0	1

2.3　減算

2.3.1　減算器

減算器は加算器と同様にして設計することが可能である．いま、被減数 M と減数 S は符号と絶対値で表示されているものとする. 始めに M と S の符号および絶対値を比較して、M − S の正しい符号が求まり、$|M| - |S|$ を求めれば良いことが判ったとする (他の場合は $|S| - |M|$ かあるいは $|M| + |S|$ を求めればよい).

減算は筆算と同様に行えば良いので、表 6 に示す真理値表が求まる.

表 6　全減算器の真理値表

M_n	S_n	B_{n-1}	D_n	B_n
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

D_n, B_n の論理式を求めると、

$$D_n = M_n \oplus S_n \oplus B_{n-1} \tag{12}$$

$$B_n = \overline{M}_n \cdot S_n + (\overline{M_n \oplus S_n}) \cdot B_{n-1} \tag{13}$$

である.

差 D_n の論理式は全加算器の和 S_n と同じ形である. しかし、借り B_n が少し異なる.

2.4 乗算

2.4.1 高速並列乗算器

被乗数 D と乗数 M は共に正の 4bit の整数であると仮定する. 図 1 は、一般形を示している. 般形の $Q_{i,j}$ と表されている項は、1bit ずつの積すなわち、$Q_{3,3} = a_3b_3, Q_{2,3} = a_2b_3, Q_{3,2} = a_3b_2$ 等を意味する. これから分かるように乗算の過程は、部分積の配列を作る過程とその部分積を加算する過程の 2 つに分けられる.N bit の数の乗算を行うとき、N^2 個の AND ゲートを用いれば、部分積の配列はゲート一段分の遅延時間で求まる。これら N^2 個の部分積の各項は同時に得られるため、組合せ回路だけで構成された一つの加算器を用いれば、中間結果をレジスタに記憶させることなく乗算を実行できる. 複数の数を同時に加算できる最も単純な加算器は図 2 に示すような組合せ回路で構成できる. この加算器は全加算器を 2 次元に並べて作られる.

$$
\begin{array}{cccccccc}
 & & & a_3 & a_2 & a_1 & a_0 & \\
 & & & b_3 & b_2 & b_1 & b_0 & \\
\hline
 & & & Q_{30} & Q_{20} & Q_{10} & Q_{00} & \\
 & & Q_{31} & Q_{21} & Q_{11} & Q_{01} & & \\
 & Q_{32} & Q_{22} & Q_{12} & Q_{02} & & & \\
Q_{33} & Q_{23} & Q_{13} & Q_{03} & & & & \\
\hline
S_7 & S_6 & S_5 & S_4 & S_3 & S_2 & S_1 & S_0
\end{array}
$$

図 1　4bit の乗算の一般形

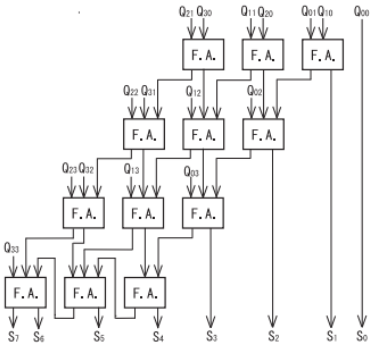

図 2　4bit 桁上げ保存加算器

2.5 除算

2.5.1 加えもどし法

被除数から除数を繰り返し引き、引き過ぎたら 1 回加算してやる方法である.2 進数の場合には 1 回引けるか否かであって、除数の 2 の補数を被除数に加えたとき符号ビットから桁上げがあれば引けたということであ

り、桁上げがなければひけなかったのであるから除数を加え元にもどしてやる。

3　実験方法

テキストにある実験課題で与えられた回路を装置を用いて作成し、正しく動作することを真理値表と照らし合わせて確認する.

4　実験結果

4.1　排他的論理和の回路を構成し、正しく動作することを確認せよ.

排他的論理和の回路を図 3 のように構成した.
真理値表は表 7 のようになった.

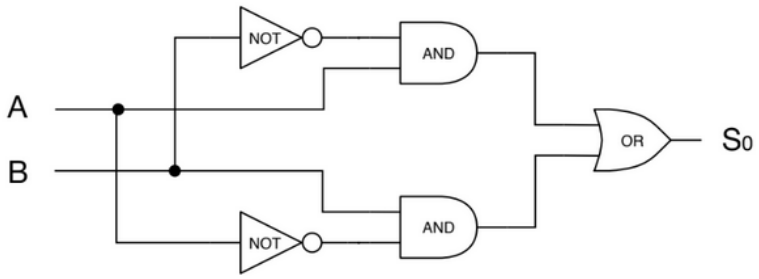

図 3　XOR の論理回路

表 7　実験課題 1 の真理値表

A	B	S_0
0	0	0
0	1	1
1	0	1
1	1	0

この結果は排他的論理和で期待していた通りの出力であり、正しく動作することを確認することができた.

4.2　半加算器を構成し、正しく動作することを確認せよ.

半加算器の回路を図 4 のように構成した.
真理値表は表 8 のようになった.
この結果は半加算器で期待していた通りの出力であり、正しく動作することを確認することができた.

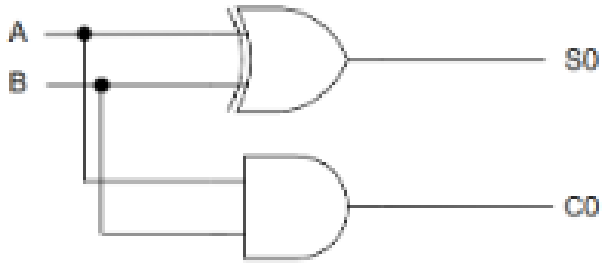

図 4　半加算器の論理回路

表 8　実験課題 2 の真理値表

A	B	S_0	C_0
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

4.3　全加算器を構成し、正しく動作することを確認せよ.

全加算器の回路を図 5 のように構成した. ここで、先程作成した半加算器を利用した. 真理値表は表 9 のようになった.

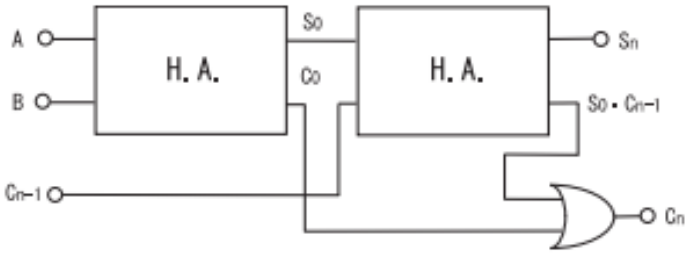

図 5　全加算器の論理回路

この結果は全加算器で期待していた通りの出力であり、正しく動作することを確認することができた.

4.4　2 ビットの全加算器を設計し、正しく動作することを確認せよ. ただし、最下位桁については半可算器を用いてよい.

2 ビットの全加算器回路を図 6 のように構成した. ここで、作成した半加算器と全加算器を利用した. 真理値表は表 10 のようになった.

この結果は 2bit の全加算器で期待していた通りの出力であり、正しく動作することを確認することができた.

表 9　実験課題 3 の真理値表

A	B	C_{n-1}	S_n	C_n
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

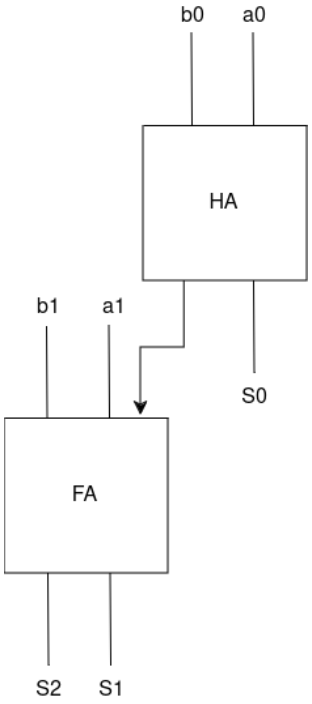

図 6　2bit 全加算器の論理回路

4.5　全減算器を設計し、正しく動作することを確認せよ.

全減算器の回路を図 7 のように構成した. 真理値表は表 11 のようになった.
この結果は減算器で期待していた通りの出力であり、正しく動作することを確認することができた.

4.6　2 ビットの減算器を設計し、正しく動作することを確認せよ.

2 ビットの減算器の回路を図 8 のように構成した. 真理値表は表 12 のようになった.

表 10　実験課題 4 の真理値表

a_0	a_1	b_0	b_1	S_0	S_1	S_2
0	0	0	0	0	0	0
0	0	0	1	0	1	0
0	0	1	0	1	0	0
0	0	1	1	1	1	0
0	1	0	0	0	1	0
0	1	0	1	0	0	1
0	1	1	0	1	1	0
0	1	1	1	1	0	1
1	0	0	0	1	0	0
1	0	0	1	1	1	0
1	0	1	0	0	1	0
1	0	1	1	0	0	1
1	1	0	0	1	1	0
1	1	0	1	1	0	1
1	1	1	0	0	0	1
1	1	1	1	0	1	1

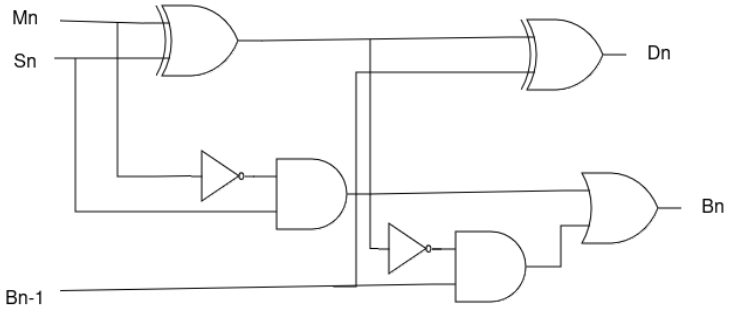

図 7　全減算器の論理回路

表 11　実験課題 5 の真理値表

M_n	S_n	B_{n-1}	D_n	B_n
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

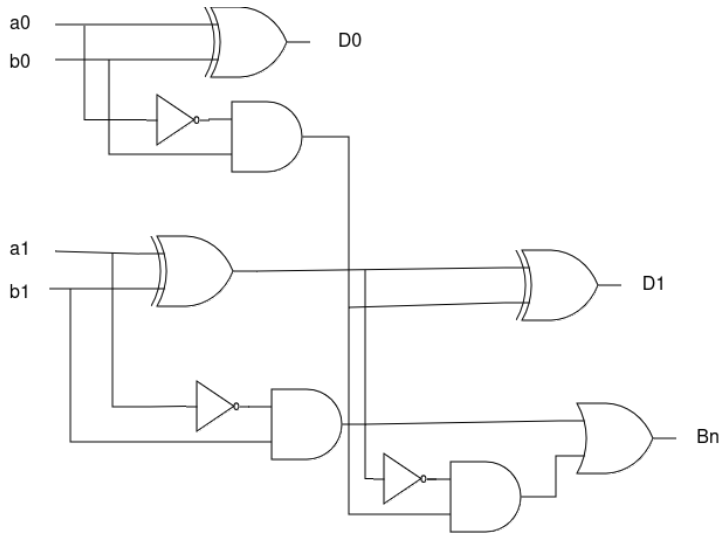

図 8　2bit 減算器の論理回路

表 12　実験課題 6 の真理値表

a_1	a_0	b_1	b_0	D_0	D_1	B_n
0	0	0	0	0	0	0
0	0	0	1	1	1	1
0	0	1	0	0	1	1
0	0	1	1	1	0	1
0	1	0	0	1	0	0
0	1	0	1	0	0	0
0	1	1	0	1	1	1
0	1	1	1	0	1	1
1	0	0	0	0	1	0
1	0	0	1	1	0	0
1	0	1	0	0	0	0
1	0	1	1	1	1	1
1	1	0	0	1	1	0
1	1	0	1	0	1	0
1	1	1	0	1	0	0
1	1	1	1	0	0	0

この結果は 2bit の減算器で期待していた通りの出力であり、正しく動作することを確認することができた.

4.7　2 ビットの乗算器を設計し、正しく動作することを確認せよ.

2 ビットの乗算器の回路を図 9 のように構成した. 真理値表は表 13 のようになった.

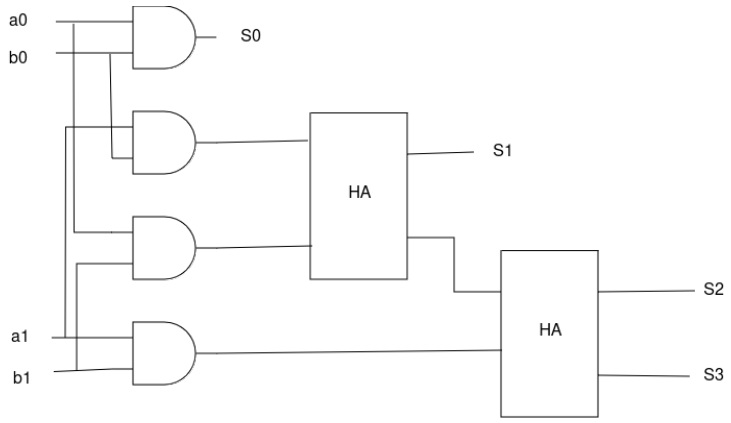

図 9　2bit 乗算器の論理回路

表 13　実験課題 7 の真理値表

a_1	a_0	b_1	b_0	S_3	S_2	S_1	S_0
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	0
0	0	1	0	0	0	0	0
0	0	1	1	0	0	0	0
0	1	0	0	0	0	0	0
0	1	0	1	0	0	0	1
0	1	1	0	0	0	1	0
0	1	1	1	0	0	1	1
1	0	0	0	0	0	0	0
1	0	0	1	0	0	1	0
1	0	1	0	0	1	0	0
1	0	1	1	0	1	1	0
1	1	0	0	0	0	0	0
1	1	0	1	0	0	1	1
1	1	1	0	0	1	1	0
1	1	1	1	1	0	0	1

この結果は 2bit の減算器で期待していた通りの出力であり、正しく動作することを確認することができた.

4.8　2 進数の除算の一例を加えもどし法で行い、レポート上で考察せよ。

例で示された $1235 \div 45$ を 2 進数で、加え戻し方によって実行する. ただし、ここでの除算は 1235 の 2 進数表現は 11bit となるから、11bit の被演算子同士で行う. ここで、1235 を 2 進数で表現すると 10011010011 となり、45 を 2 進数で表現すると 00000101101 となる. さらに、45 の 2 進数の補数表現は 11111010011 となる. 加え戻し方で除算を行うため、10011010011 に 11111010011 を桁上げがなくなるまで繰り返し足していく. 以下は Python によって生成した結果である.

1 回目: $10011010011 + 11111010011 = (1)10010100110$
2 回目: $10010100110 + 11111010011 = (1)10001111001$
3 回目: $10001111001 + 11111010011 = (1)10001001100$
4 回目: $10001001100 + 11111010011 = (1)10000011111$
5 回目: $10000011111 + 11111010011 = (1)01111110010$
6 回目: $01111110010 + 11111010011 = (1)01111000101$
7 回目: $01111000101 + 11111010011 = (1)01110011000$
8 回目: $01110011000 + 11111010011 = (1)01101101011$
9 回目: $01101101011 + 11111010011 = (1)01100111110$
10 回目: $01100111110 + 11111010011 = (1)01100010001$
11 回目: $01100010001 + 11111010011 = (1)01011100100$
12 回目: $01011100100 + 11111010011 = (1)01010110111$
13 回目: $01010110111 + 11111010011 = (1)01010001010$
14 回目: $01010001010 + 11111010011 = (1)01001011101$
15 回目: $01001011101 + 11111010011 = (1)01000110000$
16 回目: $01000110000 + 11111010011 = (1)01000000011$
17 回目: $01000000011 + 11111010011 = (1)00111010110$
18 回目: $00111010110 + 11111010011 = (1)00110101001$
19 回目: $00110101001 + 11111010011 = (1)00101111100$
20 回目: $00101111100 + 11111010011 = (1)00101001111$
21 回目: $00101001111 + 11111010011 = (1)00100100010$
22 回目: $00100100010 + 11111010011 = (1)00011110101$
23 回目: $00011110101 + 11111010011 = (1)00011001000$
24 回目: $00011001000 + 11111010011 = (1)00010011011$
25 回目: $00010011011 + 11111010011 = (1)00001101110$
26 回目: $00001101110 + 11111010011 = (1)00001000001$
27 回目: $00001000001 + 11111010011 = (1)00000010100$
28 回目: $00000010100 + 11111010011 = 11111100111$

1 回だけ余分に引いてしまっているのだから、最後の結果に 45 の 2 進数である 00000101101 を足す. $11111100111 + 00000101101 = (1)00000010100$ これは、10 進数へ変換すると 20 である. これらの結果から、$1235 \div 45 = 27 \cdots 20$ であり、さらに、正しく計算できていることが確認できた.

4.9　3 ビットの比較器を設計し、レポート上に回路図を書け.

3bit の比較 2 数 $A = A_2A_1A_0$, $B = B_2B_1B_0$ の大小関係について考える.

(i)$A = B$ となるのは、$A_2 = B_2$,$A_1 = B_1$,$A_0 = B_0$ のときのみ

(ii)$A > B$ となるのは、$A_2 > B_2$ または $A_2 = B_2, A_1 > B_1$ または $A_2 = B_2, A_1 = B_1, A_0 > B_0$

(iii)$A < B$ となるのは、$A_2 < B_2$ または $A_2 = B_2, A_1 < B_1$ または $A_2 = B_2, A_1 = B_1, A_0 < B_0$

ここで、1bit の比較器を利用する (図 10).1bit 比較器を COMP とすることにする f_0, f_1, f_2 は (1),(2),(3) と同様に定義した. これを利用して 3bit の比較器を設計した (図 11).

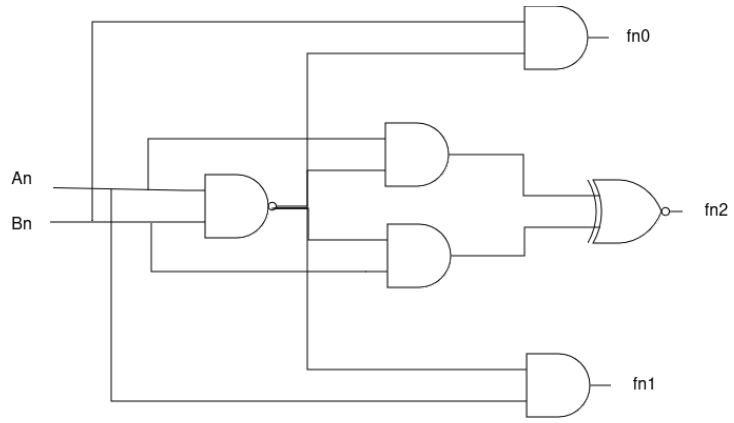

図 10　1bit 比較器の論理回路

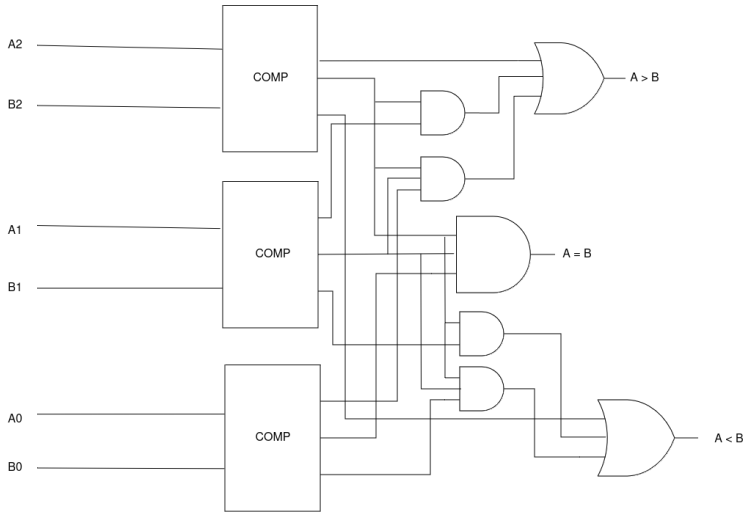

図 11　3bit 比較器の論理回路

5　考察

5.1　XOR

今回の実験では様々な回路を設計したが XOR を利用する機械が多数あった. 例として、半加算器の和の部分をあげる.0 と 0 の和は 0, 0 と 1 の和は 1,1 と 1 の和は 0 となり、XOR の論理回路における重要性について理解することができた.

5.2　加算器

半加算器は下位 bit からの繰り上げを考慮しないので足し算における 1bit 目の和を求める際に有用であることがわかった. また、全加算器は 2bit 目以降の和を求める際に必要であることも理解することができた. さらに、半加算器と全加算器を組み合わせることで 2bit 以上の加算も今回の実験課題から拡張して行えることがわかった.

5.3　減算器

減算器も加算器と同様に、半減算器と全減算器を組み合わせることによって 2bit 以上の減算器を設計することができることがわかった. また、論理関数も加算のときに比べても大きな違いはないことがわかった.

5.4　乗算器

2 ビット乗算器を実装した. 今回の乗算の方法は筆算を行うときと同様に考えるものであり、部分積を加算することで積を求めた. 部分積を加算するために加算器を利用する必要があったが. 2bit の乗算の場合には半加算器を 2 つ使うことで設計することができた. しかし、2bit よりも大きいビット数の乗算では全加算器を使う必要があることがわかった.

5.5　加え戻し法

加え戻し法は基本的に引き算と商の調整を繰り返すだけなので、手順がシンプルで理解しやすい. しかし、商の仮定と修正を繰り返すため、計算に時間がかかることがある. 特に被除数が大きな数や除数が小さい計算の場合には、繰り返しの数が多くなってしまうため効率が悪くなることがある.

5.6　比較器

加算や減算、乗算だけでなく比較においても簡単な回路の組み合わせによって回路を設計することができることがわかった. しかし、今回 3bit の比較器を設計したが入力の 1bit ごとの比較を行う必要があり、想定していた以上に回路が複雑になってしまった.

6　参考文献

[1] 群馬大学 情報科学実験 I テキスト 2024 年