1 はじめに

デジタル信号は 0 と 1 の信号の組み合わせでできている。各信号が 0 と 1 の 2 値であるが、現代のデジタル機器では幅広い数値を表現することができる。つまり離散的な値の組み合わせで、見掛け上連続的な幅を表現できるということになる。

ここでは、このコンピュータのシステムを演算装置 (ALU) から、Z80 コンピュータシステムや I/O 制御までを通して理解することを目的とする。

レポートはこの実験全体を 2 分し、以下のように各レポートで実験結果をまとめる。

- 1 部: 演算装置 (ALU:Arithmetic Logic Unit)
- 2 部: Z80 コンピュータシステム、I/O 制御

そして、このレポートはそのうちの第 1 部であり、主にデジタル論理回路の演算、またそのオーバフローについて扱う。

2 実験の目的と原理 [1]

2.1 符号ありと符号なし

n ビットのデータを扱う場合、それを 10 進変換するのに 2 つの方法がある。

1. 符号なし: n ビットをそのまま 10 進変換する方法。正の数のみ扱うことができる。
2. 符号あり: n ビットの表現範囲の約半分を負の表現に使用する方法。最上位ビットが 1 の場合は負の表現を表す。その場合は-(n ビットの 2 の補数) の絶対値として表現される。

2.2 オーバフロー

コンピュータは全て 2 進で行うが、その結果はしばしば 10 進で計算した解と異なることがある。

たとえば、

$$11000000_{(2)} + 10110000_{(2)} = 01110000_{(2)} = 112_{(10)}$$
$$192_{(10)} + 176_{(10)} = 368_{(10)}$$

というように、2 進での計算結果を 10 進に直したものと、10 進同士で計算した結果が異なる。

このように、コンピュータでの計算が実際の結果と差異を持ってしまうことをオーバフロー、という。

2.3 シフト演算

2 進のビットを左右にシフトすることに、論理演算的意義を付加することができる。主に以下のようなプロトコルがある。

1. 論理シフト演算: ビットをひとつづつ左右にずらし、はみ出たビットを捨てて、空いたビットに 0 を補填する。左シフトは 2 倍すること、右シフトは 2 で割ることに相当する。
2. 算術右シフト演算: ビットをひとつ右にずらし、はみ出したビットを捨てて、空いたビットに元の数の最上位ビットの値を補填する。

3 実験内容

1. Quartus2 で HALF_ADDER(半加算器),FULL_ADDER(全加算器),8bit_FULL_ADDER(8 ビット全加算器),8bit_SUBTRACTOR(8 ビット減算器) を設計。
2. 入力 A と入力 B に対する演算出力 O を求めた。ただし 7 つの全加算器で各ビットごとに計算を行うため、ここでは各入出力についてビットごとに 0 7(ex.A0〜A7) と表記。
3. 2 の演算結果ごとにオーバフローの有無等について考察。

4 実験結果

表 1 に ALU の演算入出力応答について事前に予想した準備課題 1 を示す。表 2 には実際の ALU の応答結果を示す。準備課題と実験課題の相違点は各表同所に*をもって示す。

これら予想と実際の結果には 2 と 3 と 4 の減算、さらに 5 で計 7 箇所の間違えがあったが、これらはすべて

- 計算ミス
- 2 進から 10 進変換、及び 10 進から 2 進変換時のミス

にまとめることができる。つまり、人的なミスを除けば予想を外れたような実験結果は生じなかった。

表 1　準備課題 1

課題	入力 A (A7～A0)		入力 B (B7～B0)		演算出力 (07～O0)		桁上がり (Cout)	桁借り (Bout)
番号	2 進表示	10 進表示	2 進表示	10 進表示	2 進表示	10 進表示	キャリー	ボロー
1(加)	00001010	10	00001010	10	00010100	20	0	×
	01010101	85	01101010	106	10111111	191	0	
	10000000	128	01000000	64	11000000	192	0	
1(減)	00011000	24	00001000	8	00010000	16	×	0
	01101001	105	00110110	54	00110011	51		0
	10000000	128	00010000	16	01110000	112		0
2(加)	10000001	129	11000000	192	01000001	65	1	×
	11000000	192	01100000	96	00100000	32	1	
	10000001	129	10000001	129	00000010	2	1	
2(減)	00001010	10	10100000	160	01101010	106	×	1
	00011110	30	01100100*	100*	10111010*	186*		1
	10100000	160	11000000	192	11100000	224		1
3(加)	00001010	10	00010101	21	00011111	31	0	×
	11110000	-16	00110000	48	00100000	32	1	
	10110000	-80	00011110	30	11001110	-50	0	
3(減)	01001001	73	00011100	28	00101101	45	×	0
	11110000	-16	11100000	-32	00010000	16		0
	10100000*	160*	11000000*	192*	11100000	-32		1
4(加)	01001001	73	01100100	100	10101101	-83	0	×
	10011000	-104	11010001	-47	01101001	105	1	
	10001000	-120	10000100	-124	00001100	12	1	
4(減)	10001000	-120	01000100	68	01000100	68	×	0
	01010000	80	10110000	-80	11100000*	-32*		1
	01001001	73	10011100	-100	10101101	-83		1
5(論)	01001010	74	00000001	1	00100101	37	×	×
	01100000	96	00000011	3	00001100	12		
	10001010	-118	00000111*	5	00000100	4		
5(算)	01101001	105	00000001	1	00110100	52	×	×
	11100011	-29	00000011	3	11111100	-4		
	10001010	-118	00000111*	5	11111100	-4		

表 2　実験課題 1(動作検証表)

課題	入力 A (A7～A0)		入力 B (B7～B0)		演算出力 (07～O0)		桁上がり (Cout)	桁借り (Bout)
番号	2 進表示	10 進表示	2 進表示	10 進表示	2 進表示	10 進表示	キャリー	ボロー
1(加)	00001010	10	00001010	10	00010100	20	0	×
	01010101	85	01101010	106	10111111	191	0	
	10000000	128	01000000	64	11000000	192	0	
1(減)	00011000	24	00001000	8	00010000	16	×	0
	01101001	105	00110110	54	00110011	51		0
	10000000	128	00010000	16	01110000	112		0
2(加)	10000001	129	11000000	192	01000001	65	1	×
	11000000	192	01100000	96	00100000	32	1	
	10000001	129	10000001	129	00000010	2	1	
2(減)	00001010	10	10100000	160	01101010	106	×	1
	00011110	30	11000000*	192*	01011110*	94*		1
	10100000	160	11000000	192	11100000	224		1
3(加)	00001010	10	00010101	21	00011111	31	0	×
	11110000	-16	00110000	48	00100000	32	1	
	10110000	-80	00011110	30	11001110	-50	0	
3(減)	01001001	73	00011100	28	00101101	45	×	0
	11110000	-16	11100000	-32	00010000	16		0
	10100000	-96*	11000000	-64*	11100000	-32		1
4(加)	01001001	73	01100100	100	10101101	-83	0	×
	10011000	-104	11010001	-47	01101001	105	1	
	10001000	-120	10000100	-124	00001100	12	1	
4(減)	10001000	-120	01000100	68	01000100	68	×	0
	01010000	80	10110000	-80	10100000*	-96*		1
	01001001	73	10011100	-100	10101101	-83		1
5(論)	01001010	74	00000001	1	00100101	37	×	×
	01100000	96	00000011	3	00001100	12		
	10001010	-118	00000101*	5	00000100	4		
5(算)	01101001	105	00000001	1	00110100	52	×	×
	11100011	-29	00000011	3	11111100	-4		
	10001010	-118	00000101*	5	11111100	-4		

5 考察

以下では、上の実験結果の検証や、それに基づきオーバフロー等に関する考察を行う。

5.1 実験結果の検証と解析

実験結果の検証と解析を行う。内容は付録を参照。

5.2 各条件におけるオーバフローの法則

以下に、符号の有無、加算と減算、に対するオーバフローの発生する法則とその理由を示す。

5.2.1 a:符号なし n ビット、加算、オーバフローを起こさない法則

- (条件)10 進出力結果が $2^n - 1$ を越えない 2 つの数同士の加算。
- (理由) 最上位ビットからの桁上がりがなく、10 進の演算結果が 2 進の表現可能な範囲に含まれているため。

5.2.2 b:符号なし n ビット、加算、オーバフローを起こす法則

- (条件)10 進出力結果が 2^n 以上となる 2 つの数同士の加算。
- (理由)10 進の演算結果が 2 進の表現可能な範囲に含まれていないため。

5.2.3 c:符号なし n ビット、減算、オーバフローを起こさない法則

- (条件)「引かれる数 A」より「引く数 B」の方が小さい。
- (理由)10 進出力が引かれる数より小さい正の数になり、2 進 n ビットで表現可能であるため。

5.2.4 d:符号なし n ビット、減算、オーバフローを起こす法則

- (条件)「引かれる数 A」より「引く数 B」の方が大きい。
- (理由)10 進出力が負の数になり、符号なし 2 進で表現で着ないため。

5.2.5 e:符号あり n ビット、加算、オーバフローを起こさない法則

- (条件)1. 異符号同士の加算。2.10 進出力結果の絶対値が $2^n - 1$ を越えない同符号同士の加算。
- (理由)1.10 進出力結果の絶対値が $2^n - 1$ を越えず、符号ビットを除く部分 (n-1 ビット) で表現で切るため。2. 符号ビットを除く n-1 ビットで $2^{n-1} - 1$ までの絶対値を表現できる

ため。

5.2.6 f:符号あり n ビット、加算、オーバフローを起こす法則

- (条件)10 進出力結果の絶対値が’2^n 以上となる同符号同士の加算。
- (理由) 最上位ビットを除く n-1 ビット数では $2^n - 1$ を越える絶対値を表現できないため。

5.2.7 g:符号あり n ビット、減算、オーバフローを起こさない法則

- (条件)1. 同符号同士の減算。2.10 進出力の絶対値が $2^{n-1} - 1$ を越えない異符号間の減算。
- (理由)1.10 進出力結果の絶対値が $2^{n-1} - 1$ を越えず、符号ビットを除く部分 (n-1 ビット) で表現で切るため。2. 符号ビットを除く n-1 ビット数で $2^{n-1} - 1$ までの絶対値を表現できるため。

5.2.8 h:符号あり n ビット、減算、オーバフローを起こす法則

- (条件)10 進出力結果の絶対値が 2^{n-1} 以上となる異符号間の減算。
- (理由) 最上位ビットを除く n01 ビットでは $2^{n-1} - 1$ を越える絶対値を表現できないため。

5.3 オーバフローの検出真理値表、論理式及びその回路

以下に、符号の有無、加算と減算、に対するオーバフローを検出する真理値表と論理式、及びその回路を示す。

5.3.1 a:符号なし、加算

- (真理値表)

表 3　符号なし、加算、のオーバフロー真理値表

ADDSUB	SIGN	COUT	BOUT	overflow
0	0	0	0	0(無)
0	0	1	0	1(有)

- (論理式)

$$overflow = COUT \tag{1}$$

- (回路図)

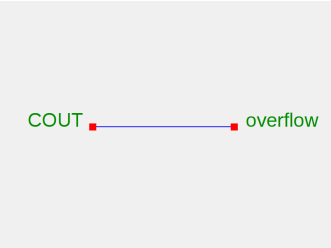

図 1　符号なし、加算、のオーバフロー検出回路

5.3.2　b:**符号なし、減算**

- (真理値表) 8 ビット ALU、加減算、オーバフロー

表 4　符号なし、減算、オーバフロー真理値表

ADDSUB	SIGN	COUT	BOUT	overflow
1	0	0	0	0
1	0	0	1	1

- (論理式)

$$overflow = BOUT \tag{2}$$

- (回路図)

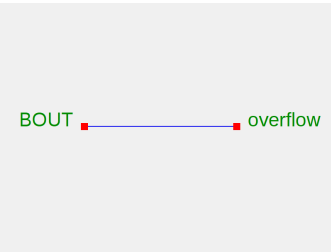

図 2　符号なし、減算、オーバフロー検出回路

5.3.3 c:**符号あり、加算**

- (真理値表)

表 5　符号あり、加算、オーバフロー真理値表

ADDSUB	SIGN	A7	B7	O7	overflow
0	1	0	0	0	0
0	1	0	0	1	1
0	1	1	1	1	0
0	1	1	1	0	1
0	1	0	1	0	0
0	1	1	0	0	0

- (論理式)

$$overflow = (\overline{A7 \bigoplus B7}) \bigwedge (A7 \bigoplus O7) \tag{3}$$

- (回路図)

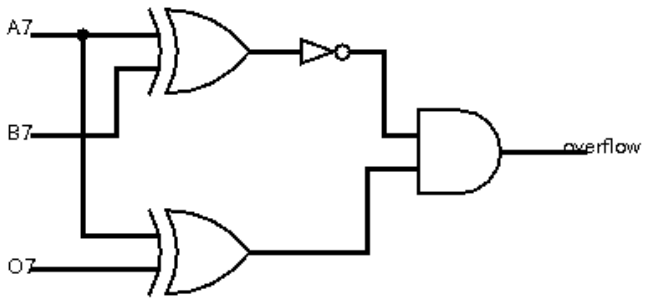

図 3　符号あり、加算、オーバフロー検出回路

5.3.4 d:**符号あり、減算**

- (真理値表)

表 6　符号あり、減算、オーバフロー真理値表

ADDSUB	SIGN	A7	B7	O7	overflow
1	1	0	0	0	0
1	1	0	0	1	0
1	1	1	1	0	0
1	1	1	1	1	0
1	1	0	1	0	0
1	1	0	1	1	1
1	1	1	0	1	0
1	1	1	0	0	1

- (論理式)

$$overflow = (A7 \bigoplus B7) \bigwedge (A7 \bigoplus O7) \tag{4}$$

- (回路図)

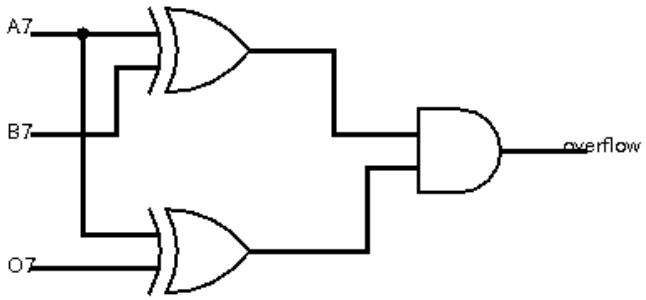

図 4　符号あり、減算、オーバフロー検出回路

5.3.5 e:8 ビット ALU、加減算

- (真理値表)

表 7　8 ビット ALU、加減算、オーバフロー真理値表

ADDSUB	SIGN	A7	B7	O7	COUT	BOUT	overflow
0	0				0	0	0(無)
0	0				1	0	1(有)
1	0				0	0	0
1	0				0	1	1
0	1	0	0	0			0
0	1	0	0	1			1
0	1	1	1	1			0
0	1	1	1	0			1
0	1	0	1	0			0
0	1	1	0	0			0
1	1	0	0	0			0
1	1	0	0	1			0
1	1	1	1	0			0
1	1	1	1	1			0
1	1	0	1	0			0
1	1	0	1	1			1
1	1	1	0	1			0
1	1	1	0	0			1

- (論理式)

$$
\begin{aligned}
overflw = {} & \overline{SIGN} \bigwedge (COUT \bigvee BOUT) \\
& \bigvee SIGN \bigwedge \overline{ADDSUB} \bigwedge (\overline{A7 \bigoplus B7}) \bigwedge (A7 \bigoplus O7) \\
& \bigvee SIGN \bigwedge ADDSUB \bigwedge (A7 \bigoplus B7) \bigwedge (A7 \bigoplus O7)
\end{aligned} \tag{5}
$$

- (回路図)

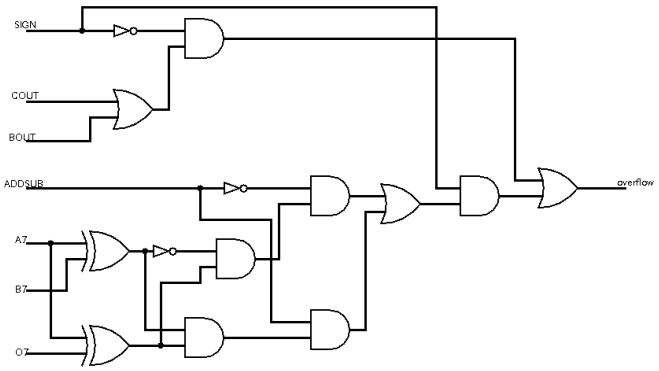

図 5　8 ビット ALU、加減算、オーバフロー検出回路

6　おわりに

今回の実験では、デジタル演算器での論理演算についてその方法とオーバフローについて主に検証した。するとデジタル演算は予測に沿った能力を持ち合わせながらも多くの例外によるオーバフローの発生も多発することがわかる。しかしそのオーバフローもある法則によって検出、制御することまで今回学んだため、計算内容によって演算プロトコルを制御すればデジタル演算には高い正確性を持たせることができる、という結果になった。

7　参考文献

[1] 雨宮好文,「現代電子回路学 [2]」, オーム社,p.131〜146