1. 目的

本実験の目的は、組み合わせ回路、演算回路、カウンタ回路などを FPGA で構成することとする。

2. 原理

2.1 FPGA

音声・画像・計測など、アナログ回路で処理されていた多くの信号処理分野でデジタル化が進んでいる。ここで、デジタル信号処理を実現するために、ロジック IC を組み合わせるか専用 LSI を開発する、もしくは CPU や DSP を使用してソフトウェアで処理するという手法が挙げられる。しかし、処理の高速化や小型化を実現しようとした場合、ロジック IC の組み合わせでは回路規模が大きくなり配線長により高速動作が困難である。一方、ソフトウェア処理は逐次処理によって実現されるため、同様に高速処理が困難である。また、専用 LSI の場合、これらの問題は解消できるものの、開発にかかる期間・費用の点で大きな問題がある。

以上の問題を解消するため、FPGA（Field Programmable Gate Array）が広く用いられている。FPGA で回路を実現する場合、配線は一つの LSI チップ上で行われるため配線長が非常に短く、高速動作可能な回路を容易に実現できる。また、ほとんどのプログラマブルな論理回路設計ツールや、CPU などをプリント基板上で組み合わせたシステムの様に後から動作内容を変更することが容易な上、FPGA は一つの LSI の中に共通のハードウェアを構成できるため、プログラムにより回路を組み合わせ（回路構成）を決定する。この仕組みにより、専用 LSI で問題となっていた開発にかかる期間や費用の点も解決することが可能になる。

FPGA の構造は大まかに分けると、I/O、ロジックセル、内部配線・スイッチ、とそれ以外の 4 つで構成されている。主にロジックセルが FPGA の中心となる部分であり、この中に論理回路を実現するためのテーブルである LUT（Look Up Table）やフリップフロップが含まれている。

また、実際の FPGA を用いた論理設計においては、HDL（Hardware Description Language）と呼ばれる言語を用いて回路構成の記述を行う。この仕組みにより、より抽象度の高いレベルでの設計が可能となり、設計期間の短縮、設計の変更・再利用が容易であるなどの利点がある。

2.2 同期式回路と非同期式回路

配線長さ等の違いによって、各回路での配線遅延が異なるため、設計したタイミングチャートに対するデータのズレは異なるタイミングとなる。これは、特に高速動作を考えるときに誤動作を引き起こす原因の一つとなる。

このタイミングずれに対して、フリップフロップで生じ得るラッチ回路を用いて入出力の状態を保持し、共通クロックによって制御することにより、この問題を解決する。こうした回路を同期式回路と呼び、FPGA においては主流となっている非同期回路に比べて広く用いられている。

次に、具体的な回路設計例としてカウンタ回路を取り上げる。カウンタは、単に計数回路としてだけでなく、タイマー回路や分周回路としての機能を持ち、モーター回転数の計測、入力が端子の状態監視、コンピュータの動作クロックの分周等、組込みシステムの中で必要不可欠なデバイスである。

図 1 に D フリップフロップを用いた非同期式 4 ビットカウンタの回路図を示す。ここで、D フリッ

プフロップはクロック端子を持った同期式フリップフロップであり、クロック端子にクロックパルスが加わるたびに入力で保持していた値を Q に出力する特性を持っている。また、反転出力.$\overline{Q}$.を入力 D にフィードバックすることによって、クロックパルスが加わるたびに出力が反転するような特性となっている。この特性を利用して、初段の出力Q_0を 2 段目のクロック端子へ入力すると、2 段目の圧力Q_1はQ_0の 1/2 分周したものが出力される。同様にQ_1を 3 段目のクロック端子、Q_2を 4 段目のクロック端子にそれぞれ入力することにより、Q_2、Q_3にはそれぞれ 1/4、1/8 分周された出力が得られる。このように、フリップフロップを n 段接続することによって2^n進カウンタを構成することが可能となる。

図 1 の構成によると、2 段目以降のフリップフロップへのクロックには、前段出力が用いられている。これは、クロック供給が直列的に与えられることを意味しており、各フリップフロップが持つ遅延によって、後段に行くほど出力の遅延が大きくなり、Q_0～Q_3のタイミングがばらつくこととなる。

図 2 に同期式 4 ビットカウンタの回路図を示す。図 1 の非同期式回路に対して、各段のフリップフロップ出力とカウンタ回路出力Q_0～Q_3との間に D フリップフロップが挿入され、共通のクロックが与えられている。これらのフリップフロップは前述のラッチ回路として動作し、共通のクロックが変化した際に同期して出力することによって、カウンタ回路に用いられるフリップフロップの出力タイミングのずれを吸収することが可能となる。

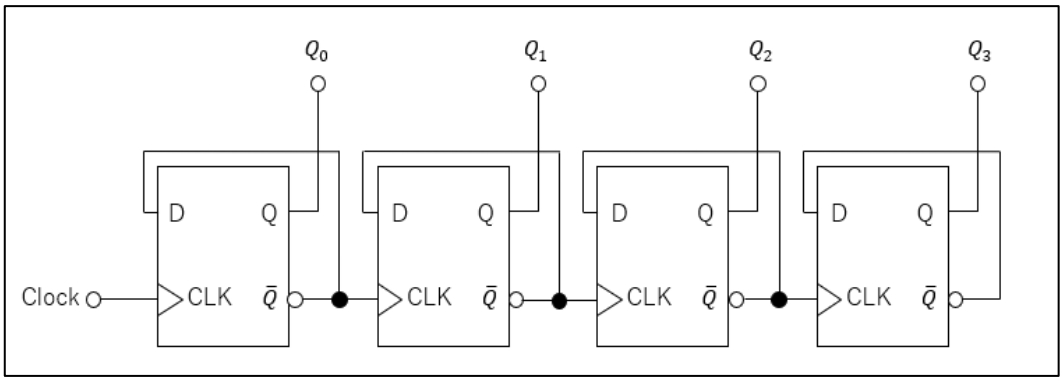

図 1　D フリップフロップによる非同期式 4 ビットカウンタ

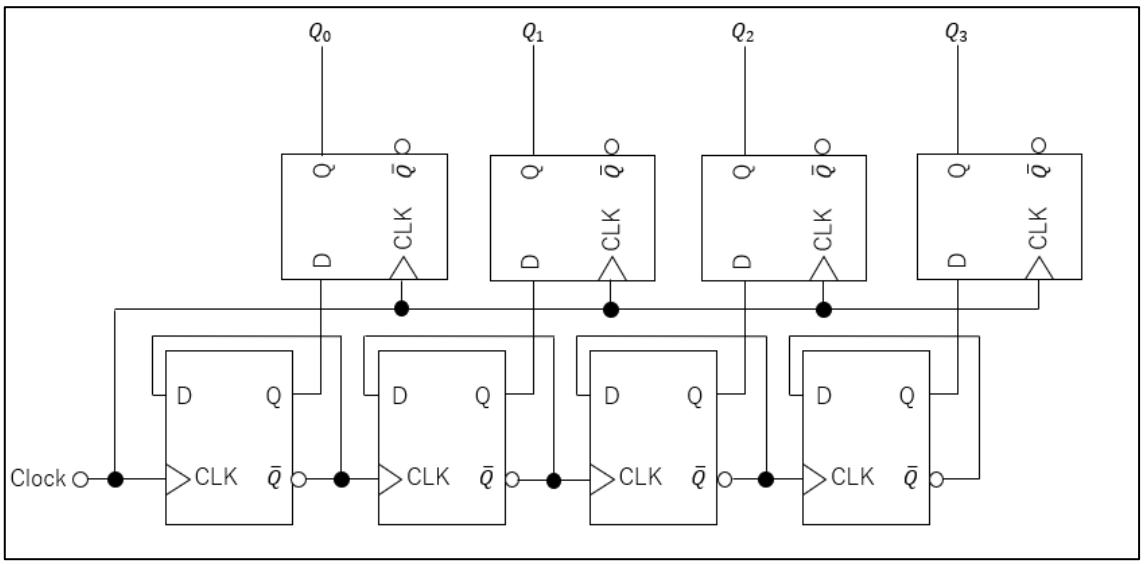

図 2　D フリップフロップによる同期式 4 ビットカウンタ

3. 方法

3.1 FPGA への実装手順

(1) パソコンと FPGA を USB ケーブルで接続した。

(2) “c:¥exp3”の各フォルダ内にある FPGA へのダウンロード用設定ファイルをダブルクリックし、[Quartus Ⅱ 13.1 Programmer]を起動した。

(3) [Hardware Setup]ボタンをクリックし、[No Hardware]となっている場合は[DE-SoC]をダブルクリックして選択した。

(4) FPGA に実装するためのオブジェクトファイルが選択されていることを確認した後、スタートボタンをクリックした。

3.2 組み合わせ回路の設計

3.2.1 特殊加法標準形による設計

(1) “c:¥exp3¥logic”内にある FPGA へのダウンロード用設定ファイルを開いた後、FPGA への実装手順に示した手順で FPGA 上に回路を構成した。

(2) 入力のスイッチを切り替えて出力を真理値表として記録した。

(3) まとめた真理値表に基づいて特殊加法標準形による論理式を作成した。組み合わせ回路の回路図を作図した。

(4) HDL のシミュレーションを行うためのソフトである Modelsim を起動するために、”c:¥exp3¥logicForModelsim”内にあるプロジェクトファイルをダブルクリックした。

(5) Modlesim を起動後、プロジェクト内にある VHDL ファイルをダブルクリックし、図 3 中の赤黒線枠内に書かれている部分の VHDL コードを真理値表に元図いて書き換え保存。

(6) [Compile]→[Compile All]と進み、VHDL ファイルをコンパイルした。

(7) コンパイルに成功したら、[Simulate]→[Start Simulation]と進んだ後、[Design Unit(s)]の欄に[work.logic]と記入し、OK ボタンを押した。

(8) シミュレーション画面に移動した後、画面下側にある[Transcript]に[do logic.do]というコマンドを入力し Enter キーを押した。この操作により、シミュレーションが実行されたため、シミュレーション波形が真理値表と一致するか確認した。

(9) 真理値表と一致するか確認した後、シミュレーション波形が表示されている wave 画面をクリックしアクティブ状態にした上で、[File]→[Export]→[Image]と進み、画像ファイルを保存した。

```
library ieee;
use ieee.std_logic_1164.all;
entity logic is
    port ( input : in std_logic_vector(2 downto 0);
           output : out std_logic);
end logic;

architecture rtl of logic is
    signal tmp : std_logic_vector(2 downto 0);
    begin
    tmp(0) <= (not input(2)) and (not input(1)) and input(0);
    tmp(1) <= (not input(2)) and input(1) and input(0);
    tmp(2) <= input(2) and input(1) and input(0);
    output <= tmp(0) or tmp(1) or tmp(2);
end rtl;
```

図 3　組み合わせ回路の VHDL コード

3.2.2　ベイチの図による組み合わせ回路の簡単化

(1)　先に設計した組み合わせ回路について、特殊加法標準形による設計(2)の真理値表からベイチの図を作図し、簡単化した。ベイチの図をもとに、論理式を作成し、回路図を作図した。

(2)　作成した回路図に基づき、特殊加法標準形による設計(4)-(9)を繰り返した。また、図 3 中の黒線枠内に書かれていた VHDL コードは図 4 のように書き換えた。

```
tmp(0) <= (not input(2)) and input(0);

tmp(1) <= input(1) and input(0);

output <= tmp(0) or tmp(1);
```

図 4　簡単化した組み合わせ回路の VHDL コード（一部抜粋）

3.3　演算回路の設計

3.3.1　FA（Full Adder）の設計

(1)　“c:¥exp3¥fa” 内にある FPGA へのダウンロード用設定ファイルを開いた後、FPGA への実装手順に示した手順で FPGA 上に回路を構成した。

(2)　入力のスイッチ（入力 A : SW6、入力 B : SW2、桁上げ入力 Cin : KEY3）を切り替えて、LED 出力（和 Sum : LEDR1、桁上げ出力 Cout : LEDR0）を確認し、記録した。正しい FA 出力が得られていることを確認した。

3.3.2　4 ビット加算器の設計

(1)　“c:¥exp3¥fa4” 内にある FPGA へのダウンロード用設定ファイルを開いた後、FPGA への実装手順に示した手順で FPGA 上に 4 ビット加算器を構成した。

(2)　入力のスイッチ（入力 A : SW9~SW6、入力 B : SW5~SW2）を切り替えて、7 セグメント LED および桁上げ出力 Cout（LEDR0）を確認し、記録した。正しい加算出力が得られていることを確認した。

3.3.3　ALU（Arithmetic Logic Unit）の設計

(1) “c:¥exp3¥alu” 内にある FPGA へのダウンロード用設定ファイルを開いた後、FPGA への実装手順に示した手順で FPGA 上に図 5 に示す ALU 回路を構成した。

(2) 入力のスイッチ（入力 A：SW9~SW6、入力 B：SW5~SW2）を任意の値に設定した状態で選択信号（S_1：SW1、S_0：SW0、C_0：KEY3）を網羅的に切り替え、7 セグメント LED および桁上げ出力 Cout（LEDR0）を確認し、記録した。正しい出力が得られていることを確認した。

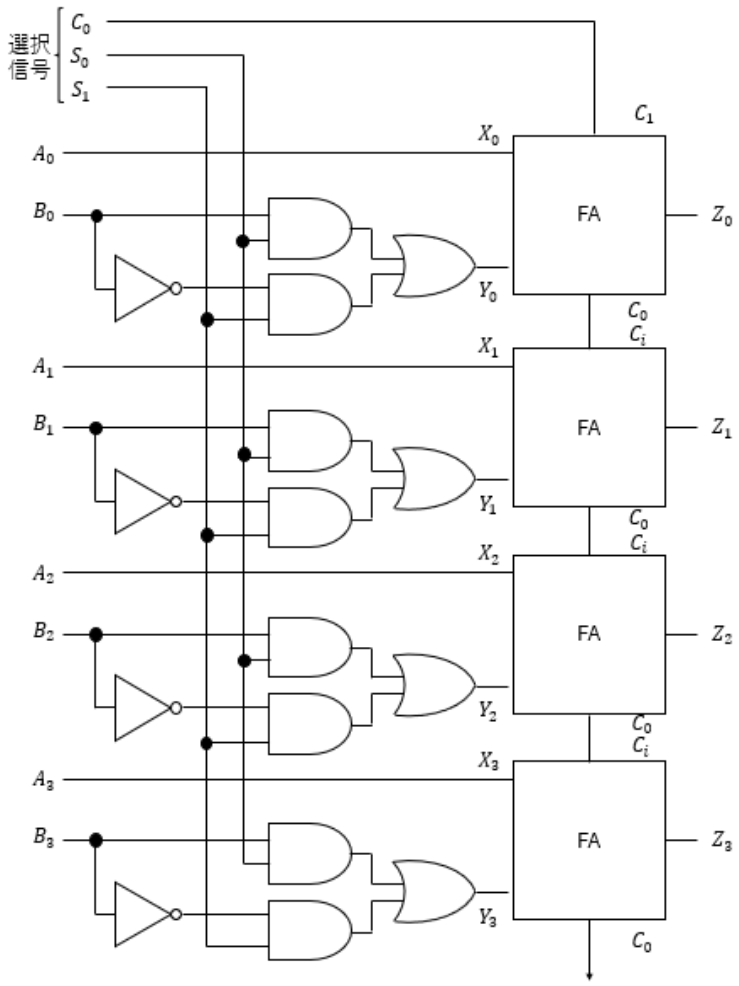

図 5　ALU 回路

3.4　カウンタ回路の設計

3.4.1　非同期式カウンタ回路の設計

(1) FPGA の GPIO 0 の 1～8 番ピンに、ロジックアナライザの 0～7 番を接続した。また、ロジックアナライザのグラウンド線（黒）は GPIO 0 の 12 番ピンに接続した。

(2) “c:¥exp3¥cnt4bit”内にある FPGA へのダウンロード用設定ファイルを開いた後、FPGA への実装手順に示した手順で FPGA 上に非同期式カウンタ回路を構成した。

(3) ロジックアナライザの[Save/Recall]ボタンを押した後、[リコール]ボタンを押し、[呼出:セットアップ]であることを確認した上で、[ロード元:setup_0]を選択した。1 周期分のカウント値が見える

ように、ロジックアナライザの時間レンジを調整した。

(4) ロジックアナライザの[Single]ボタンを押すことで波形を止め、出力された波形を画像形式で保存した。

(5) 時間レンジを最小とした後、表示されている波形の立ち下がる場所を検索し、各カウントの立ち下がり時間幅を[ns]単位で測定した。

3.4.2 同期式カウンタ回路の設計

(1) 配線は非同期式カウンタ回路の設計の時の回路を用いた。

(2) “c:¥exp3¥cnt4bit_sync”内にある FPGA へのダウンロード用設定ファイルを開いた後、FPGA への実装手順で示した手順で FPGA 上に同期式カウンタ回路を構成した。

(3) 非同期式カウンタ回路の設計(3)～(5)の作業を繰り返した。

3.5 PWM (Pulse Width Modulation) 発生回路の設計

(1) 配線は非同期式カウンタ回路の設計の時の回路を用いた。

(2) “c:¥exp3¥pwm”内にある FPGA へのダウンロード用設定ファイルを開いた後、FPGA への実装手順に示した手順で、図 2 の同期式カウンタ回路と図 5 の ALU 回路を用いた PWM 発生回路を FPGA 上に構成した。

(3) スイッチ（SW9～SW6）によって Duty 比を変化させ、ロジックアナライザで幾つかの PWM 出力波形を保存した。

(4) ロジックアナライザの全ての配線を外した後、PWM 出力とグランドにテスタを接続した。そして Duty 比を変化させたときの PWM 出力電源を測定し、記録した。

4. 予習課題

(1) FPGA とマイコンの違いをまとめる

マイコンとは、ソフトウェアを変更することでハードウェアの動作を行う機械であり、用途はマイコン自体のハードウェアに絞られる。FPGA ではハードウェア自体のプログラムによって変更できるため、マイコンより柔軟性に優れている[2]。よって、状況により、行いたい動作が異なる場合や、現場で動作を変更したい場合などは FPGA を用いると良いと考えられる。

(2) CPU、ALU、メモリ、FF、NAND、トランジスタ、半導体の関係を簡潔にまとめる

CPU とは、ALU、トランジスタ、またそれらを制御するユニットから構成されている。また、ALU は論理回路と FF で構成されている。FF はメモリやトランジスタを用いて構成する。NAND 回路はトランジスタを組み合わせて構成されている。トランジスタは半導体を用いて作成されている。よって、これらの関係は全て CPU 内に存在する物を大まかに分けてものだと考えることができる。また、それら全てに半導体が使われていることが分かった。

(3) D-FF の真理値表と動作について示す

D-FF とは D フリップフロップのことである。D-FF はクロック信号の立ち上がりや立ち下がりに同

期して、出力を変化させるフリップフロップである[1]。真理値表は以下の表 1 に示す。

表 1　D-FF の真理値表

入力値	出力値	
D	Q	$\overline{Q}$
0	0	1
1	1	0

動作は、加えられる入力を記憶し、そのまま出力に出すことで、入力を遅延させている。これによって、クロック信号を用いる時には、前の時刻の入力を、次の時刻まで保持するような動作となる。[1] また、クロック信号を用いる際には、立ち上がりエッジトリガ、立ち下がりエッジトリガがある。立ちあがりでは、0 から 1 になったタイミングで入力を更新し、立ち下がりでは逆に 1 から 0 になったタイミングで入力を更新する。

(4)　図 1 の非同期式カウンタと図 2 の同期式カウンタについて、入力クロックに対する出力Q_0～Q_3を示すタイムチャートを描く

図 1 の非同期式カウンタの入力クロックに対する出力を示すタイムチャートを図 6 に示す。また、今回のタイムチャートでは立ち上がり方式を採用している。

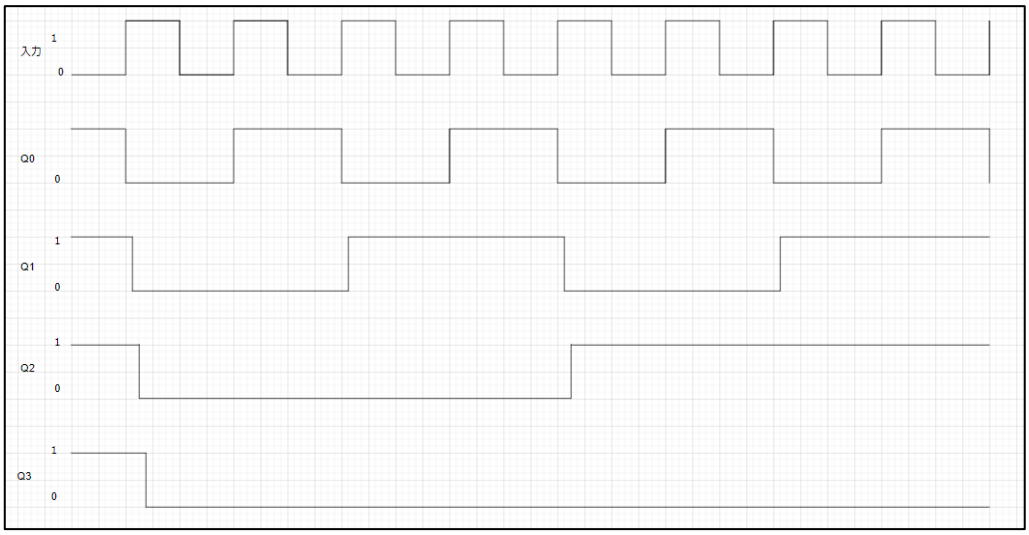

図 6　非同期式カウンタのタイムチャート

図 6 では、クロックが 1 になったタイミングの入力を更新している。最初に 1 になったタイミングで Q0 が 0 であったことから、それ以降も遅延があるが 0 になっている。また、遅延のせいで全体的にずれており、クロックの動作とズレが生じている出力がされている。

図 2 の同期式カウンタの入力クロックに対する出力を示すタイムチャートを図 7 に示す。

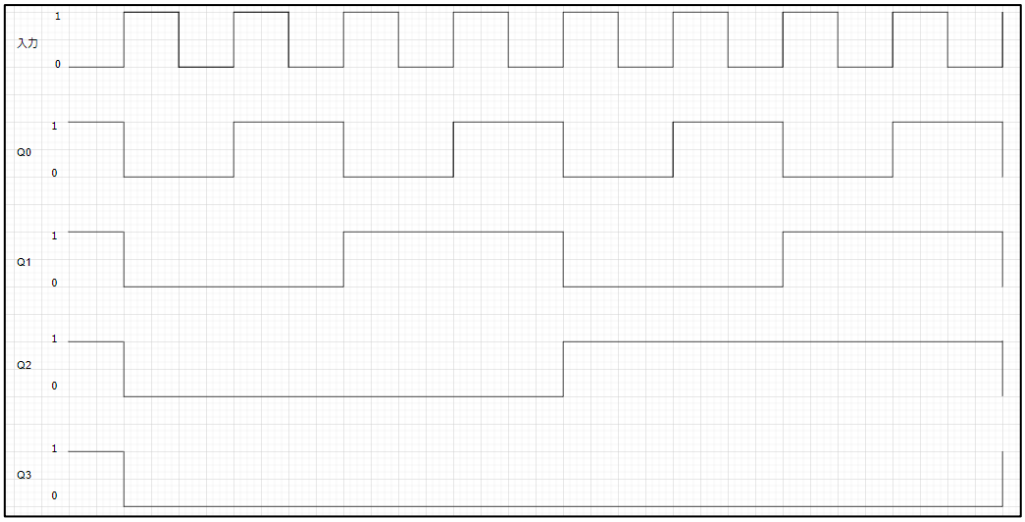

図 7　同期式カウンタのタイムチャート

図 7 では非同期式カウンタの遅延がなくなったタイムチャートになっており、クロックの動作と完全に一致するような動作をしており、最終的な Q3 がきちんと出力が 1 になっていることが確認することができる。

(5)　図 5 の ALU 回路は 4 ビットの 2 進数 A と B を操作し、出力 Z を得る回路である。選択信号C_0、S_0、S_1を変化させたときの出力 Z に関する真理値表を作成する

今回、$\bar{B}$に関しては記入の関係上 -B と記入を行っている。また、C の値は A や B の値が決まらない限りわからないため、省いている。今回はそれぞれ Z の出力を真理値表に表した。表 2 に真理値表を示す。

表 2　ALU 回路の出力 Z の真理値表

C0	S1	S0	X	Y	Z
0	0	0	A	0	A
0	0	1	A	B	A+B
0	1	0	A	-B	A+(-B)
0	1	1	A	1	A+1
1	0	0	A	0	A+1
1	0	1	A	B	A+B+1
1	1	0	A	-B	A+(-B)+1
1	1	1	A	1	A

(6)　PWM の概要や特徴および用途について示す。また、図 2 の同期式カウンタ回路と図 5 の ALU 回路を用いて PWM 発生回路を設計する

PWM とは、0 か 5 しかない直流電圧を状況によって制御するための手法である。電圧をパルスとして用いることで、直流電圧の値がパルスの平均値としてみなすことができるため、それを用いて電圧制御を行っている。また、それらを表す Duty 比というものが存在し、high の時間 / パルスの周期 のこ

とを表す。用途としては、直流モーターの速度制御などに使われている。

回路を用いて、PWM 発生装置を設計する際には、Duty 比を 6.25%（1/16）刻みで変更できるようにする。実際に作成した回路を図 8 に示す。

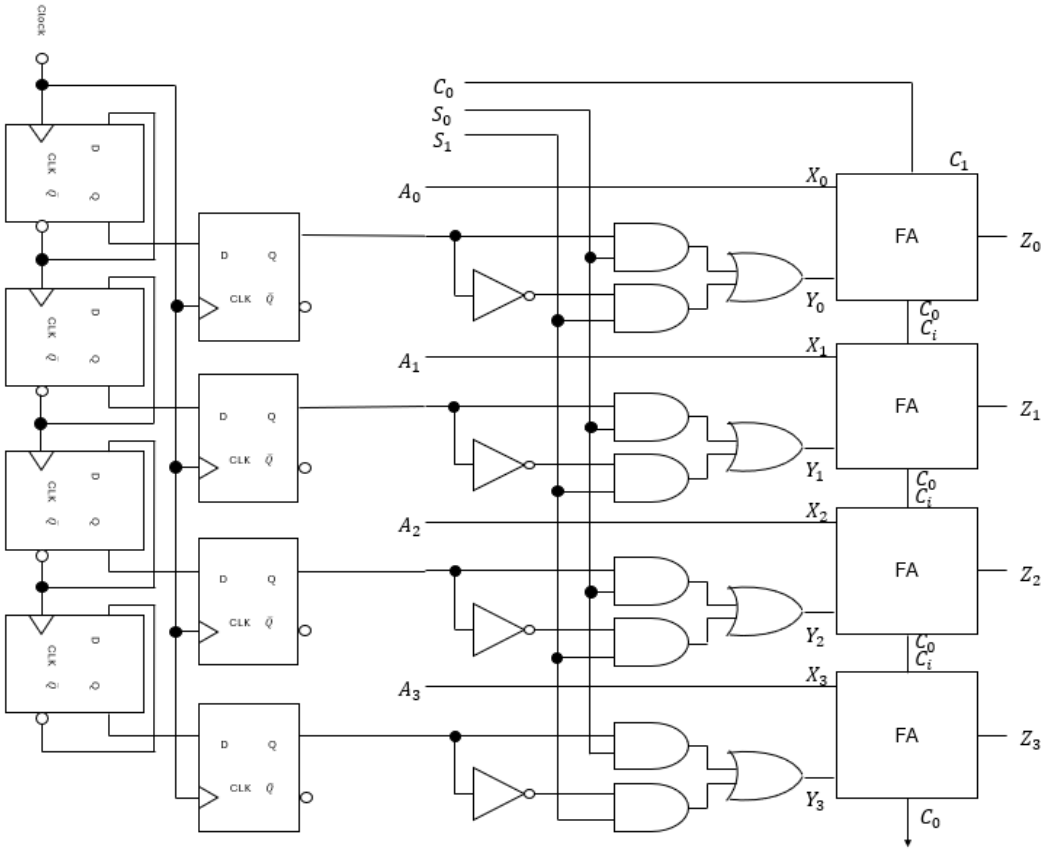

図 8　同期式カウンタと ALU 回路を用いた PWM 発生回路

図 8 では ALU 回路の B の入力を行っていたところに同期式カウンタの出力を入れており、それによって接続を行っている。また、A の入力では Duty 比の値を固定値として流す。最終的に Z が出力されるが、この Z の値と、同期式カウンタの出力値によって、PWM を発生させることができる。

5. 使用器具

表 3 に本実験で使用した器具を示す。

表 3　使用器具

名称	製造会社	製造番号	備品番号	形式
パソコン	-	-	CST-16-8140	-
FPGA	-	14100016-1449	-	-
オシロスコープ	-	MY54020241	2013-171-0000965	-
テスター	Agilent	MY54280046	2015-171-0000205	-

6. 実験結果

6.1　組み合わせ回路の設計

6.1.1　特殊加法標準形による設計

実験で用いたスイッチを変更した組み合わせ回路の真理値表を表 4 に示す。

表 4　組み合わせ回路の真理値表

SW2	SW1	SW0	Z
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

表 4 の真理値表を用いて、特殊加法標準形により論理式を作成した。結果を下記に示す。

$$OUT = \overline{SW0}SW1\overline{SW2} + SW0SW1\overline{SW2} + SW0SW1SW2 \quad (1)$$

(1)式を回路図にしたものを図 9 に示す。

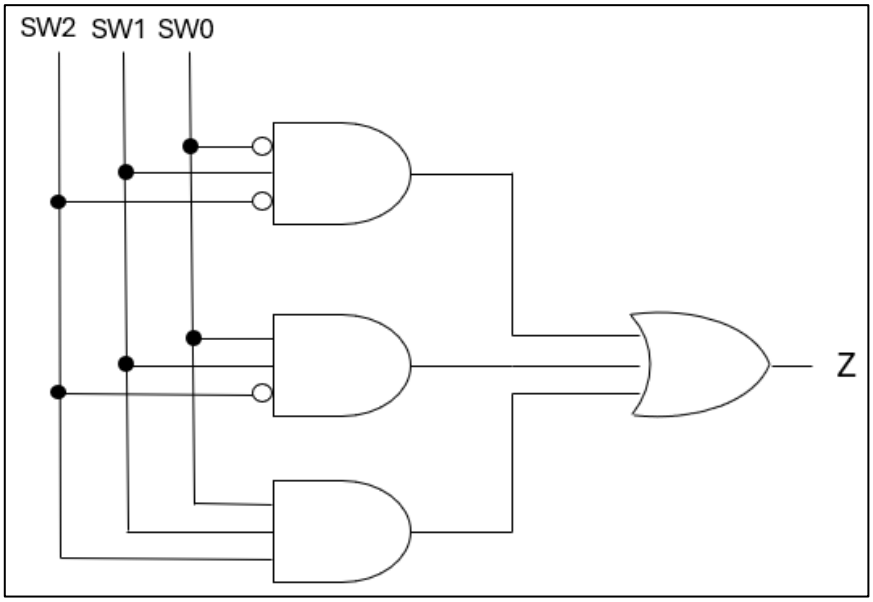

図 9　組み合わせ回路の回路図

(1)式となるようにコードを書き換えたものを図 10 に示す。

```
library ieee;
use ieee.std_logic_1164.all;

entity logic is
        port ( input : in std_logic_vector(2 downto 0);
                output : out std_logic);
end logic;

architecture rtl of logic is
        signal tmp:std_logic_vector(2 downto 0);
        begin
        tmp(0)<= (not input(2)) and (input(1)) and (not input(0));
        tmp(1)<= (not input(2)) and(input(1)) and (input(0));
        tmp(2)<= (input(2)) and (input(1)) and (input(0));
        output <= tmp(2) or tmp(1) or tmp(0);

end rtl;
```

図 10　(1)の論理式のコード

図 10 のコードを用いてシミュレーションを行った結果を図 11 に示す。また、長さの関係上 3 つに分割を行っている。

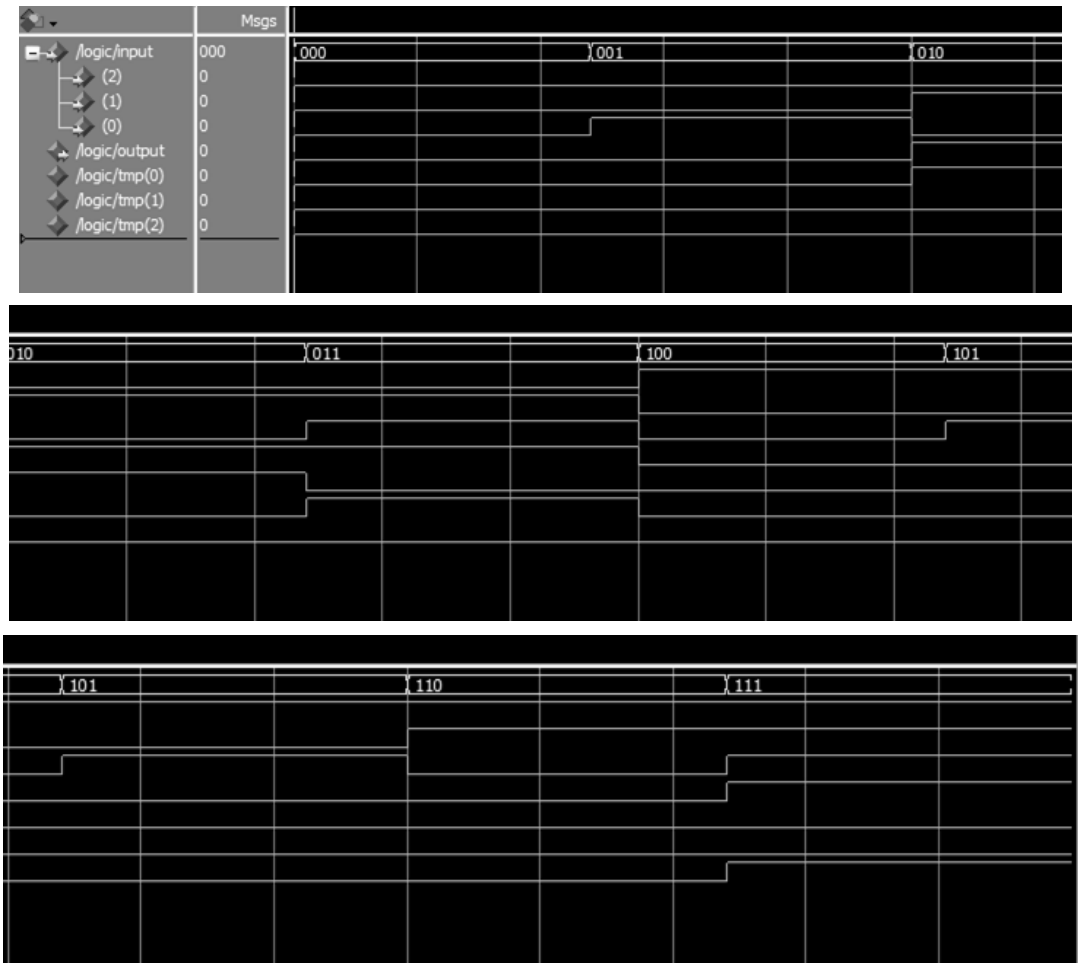

図 11　図 10 のコードのシミュレーション結果

図 11 では出力が 010、011、111 の時に 1 になっていることがわかる。

6.1.2　ベイチの図による組み合わせ回路の簡単化

表 4 の真理値表を元にベイチの図を作成した。図 12 に示す。

		SW0		
	1	1		
		1		SW2
	SW1			

図 12　組み合わせ回路の真理値表のベイチ図

図 12 のベイチ図を元に簡単化を行った論理式を作成した。結果を下記に示す。

$$OUT = SW1\overline{SW2} + SW0SW1 \quad (2)$$

(2)式を元に回路図を作成した。図 13 に示す。

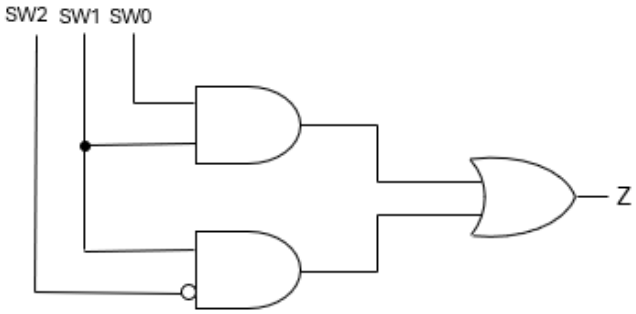

図 13　簡単化を行った論理式の回路図

(2)式を元にコードを書き直した結果を図 14 に示す。

```
library ieee;
use ieee.std_logic_1164.all;

entity logic is
        port ( input : in std_logic_vector(2 downto 0);
                output : out std_logic);
end logic;

architecture rtl of logic is
        signal tmp:std_logic_vector(2 downto 0);
        begin
        tmp(0)<= (not input(2)) and (input(1));
        tmp(1)<= (input(1)) and (input(0));
        output <= tmp(1) or tmp(0);

end rtl;
```

図 14　簡単化した論理式を用いたコード

図 14 を実行したシミュレーション結果を図 15 に示す。また、長さの関係上 3 分割している。

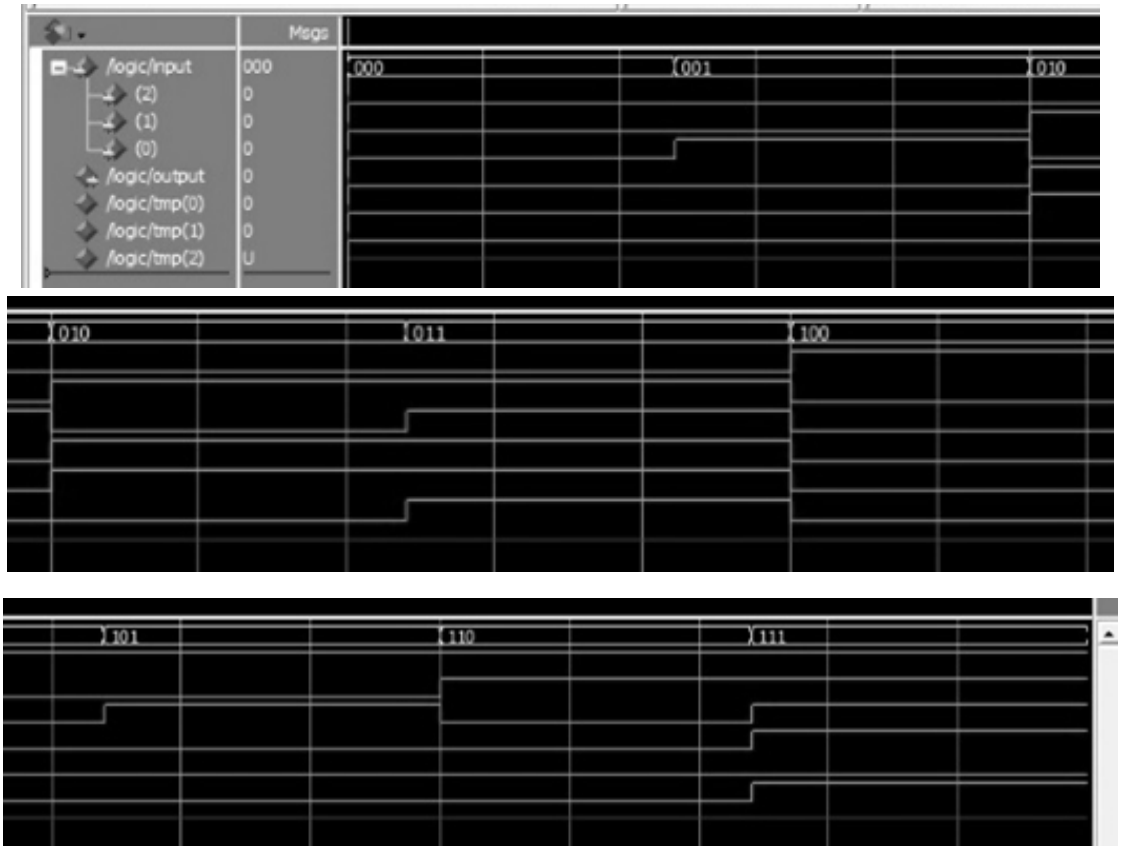

図 15　簡単化した論理式のコードのシミュレーション結果

結果として、出力が 010、011、111 の時に 1 になっていることがわかり、簡単化が成功していることがわかる。

6.2　演算回路の設計

6.2.1　FA の設計

FA の設計を行い、動作を確認した。真理値表を表 5 に示す。

表 5　FA 回路の動作確認をした真理値表

A(SW6)	B(SW2)	C(KEY3)	C(LEDR0)	Z(LEDR1)
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

結果として、FA の真理値表と一緒になっていることがわかる

6.2.2　4 ビット加算器の設計

4 ビット加算器の設計を行い、A と B を網羅的に変更し、出力結果を確認した。真理値表を表 6、7、8、9、10、11、12 に示す。また、長いため分割を行っている

表 6　4 ビット加算器の設計(1/7)

A				B				結果	
SW9	SW8	SW7	SW6	SW5	SW4	SW3	SW2	LED	C
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1	1	0
0	0	0	0	0	0	1	0	2	0
0	0	0	0	0	0	1	1	3	0
0	0	0	0	0	1	0	0	4	0
0	0	0	0	0	1	0	1	5	0
0	0	0	0	0	1	1	0	6	0
0	0	0	0	0	1	1	1	7	0
0	0	0	0	1	0	0	0	8	0
0	0	0	0	1	0	0	1	9	0
0	0	0	0	1	0	1	0	A	0
0	0	0	0	1	0	1	1	B	0
0	0	0	0	1	1	0	0	C	0
0	0	0	0	1	1	0	1	D	0
0	0	0	0	1	1	1	0	E	0
0	0	0	0	1	1	1	1	F	0
0	0	0	1	0	0	0	0	1	0
0	0	0	1	0	0	0	1	2	0
0	0	0	1	0	0	1	0	3	0
0	0	0	1	0	0	1	1	4	0
0	0	0	1	0	1	0	0	5	0
0	0	0	1	0	1	0	1	6	0
0	0	0	1	0	1	1	0	7	0
0	0	0	1	0	1	1	1	8	0
0	0	0	1	1	0	0	0	9	0
0	0	0	1	1	0	0	1	A	0
0	0	0	1	1	0	1	0	B	0
0	0	0	1	1	0	1	1	C	0
0	0	0	1	1	1	0	0	D	0
0	0	0	1	1	1	0	1	E	0
0	0	0	1	1	1	1	0	F	0
0	0	0	1	1	1	1	1	0	1
0	0	1	0	0	0	0	0	2	0
0	0	1	0	0	0	0	1	3	0
0	0	1	0	0	0	1	0	4	0
0	0	1	0	0	0	1	1	5	0
0	0	1	0	0	1	0	0	6	0
0	0	1	0	0	1	0	1	7	0
0	0	1	0	0	1	1	0	8	0
0	0	1	0	0	1	1	1	9	0

表 7　4 ビット加算器の設計(2/7)

A				B				結果	
SW9	SW8	SW7	SW6	SW5	SW4	SW3	SW2	LED	C
0	0	1	0	1	0	0	0	A	0
0	0	1	0	1	0	0	1	B	0
0	0	1	0	1	0	1	0	C	0
0	0	1	0	1	0	1	1	D	0
0	0	1	0	1	1	0	0	E	0
0	0	1	0	1	1	0	1	F	0
0	0	1	0	1	1	1	0	0	1
0	0	1	0	1	1	1	1	1	0
0	0	1	1	0	0	0	0	3	0
0	0	1	1	0	0	0	1	4	0
0	0	1	1	0	0	1	0	5	0
0	0	1	1	0	0	1	1	6	0
0	0	1	1	0	1	0	0	7	0
0	0	1	1	0	1	0	1	8	0
0	0	1	1	0	1	1	0	9	0
0	0	1	1	0	1	1	1	A	0
0	0	1	1	1	0	0	0	B	0
0	0	1	1	1	0	0	1	C	0
0	0	1	1	1	0	1	0	D	0
0	0	1	1	1	0	1	1	E	0
0	0	1	1	1	1	0	0	F	0
0	0	1	1	1	1	0	1	0	1
0	0	1	1	1	1	1	0	1	1
0	0	1	1	1	1	1	1	2	1
0	1	0	0	0	0	0	0	4	0
0	1	0	0	0	0	0	1	5	0
0	1	0	0	0	0	1	0	6	0
0	1	0	0	0	0	1	1	7	0
0	1	0	0	0	1	0	0	8	0
0	1	0	0	0	1	0	1	9	0
0	1	0	0	0	1	1	0	A	0
0	1	0	0	0	1	1	1	B	0
0	1	0	0	1	0	0	0	C	0
0	1	0	0	1	0	0	1	D	0
0	1	0	0	1	0	1	0	E	0
0	1	0	0	1	0	1	1	F	0
0	1	0	0	1	1	0	0	0	1
0	1	0	0	1	1	0	1	1	1
0	1	0	0	1	1	1	0	2	1
0	1	0	0	1	1	1	1	3	1

表 8　4 ビット加算器の設計(3/7)

A				B				結果	
SW9	SW8	SW7	SW6	SW5	SW4	SW3	SW2	LED	C
0	1	0	1	0	0	0	0	5	0
0	1	0	1	0	0	0	1	6	0
0	1	0	1	0	0	1	0	7	0
0	1	0	1	0	0	1	1	8	0
0	1	0	1	0	1	0	0	9	0
0	1	0	1	0	1	0	1	A	0
0	1	0	1	0	1	1	0	B	0
0	1	0	1	0	1	1	1	C	0
0	1	0	1	1	0	0	0	D	0
0	1	0	1	1	0	0	1	E	0
0	1	0	1	1	0	1	0	F	0
0	1	0	1	1	0	1	1	0	1
0	1	0	1	1	1	0	0	1	1
0	1	0	1	1	1	0	1	2	1
0	1	0	1	1	1	1	0	3	1
0	1	0	1	1	1	1	1	4	1
0	1	1	0	0	0	0	0	6	0
0	1	1	0	0	0	0	1	7	0
0	1	1	0	0	0	1	0	8	0
0	1	1	0	0	0	1	1	9	0
0	1	1	0	0	1	0	0	A	0
0	1	1	0	0	1	0	1	B	0
0	1	1	0	0	1	1	0	C	0
0	1	1	0	0	1	1	1	D	0
0	1	1	0	1	0	0	0	E	0
0	1	1	0	1	0	0	1	F	0
0	1	1	0	1	0	1	0	0	1
0	1	1	0	1	0	1	1	1	1
0	1	1	0	1	1	0	0	2	1
0	1	1	0	1	1	0	1	3	1
0	1	1	0	1	1	1	0	4	1
0	1	1	0	1	1	1	1	5	1
0	1	1	1	0	0	0	0	7	0
0	1	1	1	0	0	0	1	8	0
0	1	1	1	0	0	1	0	9	0
0	1	1	1	0	0	1	1	A	0
0	1	1	1	0	1	0	0	B	0
0	1	1	1	0	1	0	1	C	0
0	1	1	1	0	1	1	0	D	0
0	1	1	1	0	1	1	1	E	0

表 9　4 ビット加算器の設計(4/7)

A				B				結果	
SW9	SW8	SW7	SW6	SW5	SW4	SW3	SW2	LED	C
0	1	1	1	1	0	0	0	F	0
0	1	1	1	1	0	0	1	0	1
0	1	1	1	1	0	1	0	1	1
0	1	1	1	1	0	1	1	2	1
0	1	1	1	1	1	0	0	3	1
0	1	1	1	1	1	0	1	4	1
0	1	1	1	1	1	1	0	5	1
0	1	1	1	1	1	1	1	6	1
1	0	0	0	0	0	0	0	8	0
1	0	0	0	0	0	0	1	9	0
1	0	0	0	0	0	1	0	A	0
1	0	0	0	0	0	1	1	B	0
1	0	0	0	0	1	0	0	C	0
1	0	0	0	0	1	0	1	D	0
1	0	0	0	0	1	1	0	E	0
1	0	0	0	0	1	1	1	F	0
1	0	0	0	1	0	0	0	0	1
1	0	0	0	1	0	0	1	1	1
1	0	0	0	1	0	1	0	2	1
1	0	0	0	1	0	1	1	3	1
1	0	0	0	1	1	0	0	4	1
1	0	0	0	1	1	0	1	5	1
1	0	0	0	1	1	1	0	6	1
1	0	0	0	1	1	1	1	7	1
1	0	0	1	0	0	0	0	9	0
1	0	0	1	0	0	0	1	A	0
1	0	0	1	0	0	1	0	B	0
1	0	0	1	0	0	1	1	C	0
1	0	0	1	0	1	0	0	D	0
1	0	0	1	0	1	0	1	E	0
1	0	0	1	0	1	1	0	F	0
1	0	0	1	0	1	1	1	0	1
1	0	0	1	1	0	0	0	1	1
1	0	0	1	1	0	0	1	2	1
1	0	0	1	1	0	1	0	3	1
1	0	0	1	1	0	1	1	4	1
1	0	0	1	1	1	0	0	5	1
1	0	0	1	1	1	0	1	6	1
1	0	0	1	1	1	1	0	7	1
1	0	0	1	1	1	1	1	8	1

表 10　4 ビット加算器の設計(5/7)

A				B				結果	
SW9	SW8	SW7	SW6	SW5	SW4	SW3	SW2	LED	C
1	0	1	0	0	0	0	0	A	0
1	0	1	0	0	0	0	1	B	0
1	0	1	0	0	0	1	0	C	0
1	0	1	0	0	0	1	1	D	0
1	0	1	0	0	1	0	0	E	0
1	0	1	0	0	1	0	1	F	0
1	0	1	0	0	1	1	0	0	1
1	0	1	0	0	1	1	1	1	1
1	0	1	0	1	0	0	0	2	1
1	0	1	0	1	0	0	1	3	1
1	0	1	0	1	0	1	0	4	1
1	0	1	0	1	0	1	1	5	1
1	0	1	0	1	1	0	0	6	1
1	0	1	0	1	1	0	1	7	1
1	0	1	0	1	1	1	0	8	1
1	0	1	0	1	1	1	1	9	1
1	0	1	1	0	0	0	0	B	0
1	0	1	1	0	0	0	1	C	0
1	0	1	1	0	0	1	0	D	0
1	0	1	1	0	0	1	1	E	0
1	0	1	1	0	1	0	0	F	0
1	0	1	1	0	1	0	1	0	1
1	0	1	1	0	1	1	0	1	1
1	0	1	1	0	1	1	1	2	1
1	0	1	1	1	0	0	0	3	1
1	0	1	1	1	0	0	1	4	1
1	0	1	1	1	0	1	0	5	1
1	0	1	1	1	0	1	1	6	1
1	0	1	1	1	1	0	0	7	1
1	0	1	1	1	1	0	1	8	1
1	0	1	1	1	1	1	0	9	1
1	0	1	1	1	1	1	1	A	1
1	1	0	0	0	0	0	0	C	0
1	1	0	0	0	0	0	1	D	0
1	1	0	0	0	0	1	0	E	0
1	1	0	0	0	0	1	1	F	0
1	1	0	0	0	1	0	0	0	1
1	1	0	0	0	1	0	1	1	1
1	1	0	0	0	1	1	0	2	1
1	1	0	0	0	1	1	1	3	1

表 11　4 ビット加算器の設計(6/7)

A				B				結果	
SW9	SW8	SW7	SW6	SW5	SW4	SW3	SW2	LED	C
1	1	0	0	1	0	0	0	4	1
1	1	0	0	1	0	0	1	5	1
1	1	0	0	1	0	1	0	6	1
1	1	0	0	1	0	1	1	7	1
1	1	0	0	1	1	0	0	8	1
1	1	0	0	1	1	0	1	9	1
1	1	0	0	1	1	1	0	A	1
1	1	0	0	1	1	1	1	B	1
1	1	0	1	0	0	0	0	D	0
1	1	0	1	0	0	0	1	E	0
1	1	0	1	0	0	1	0	F	0
1	1	0	1	0	0	1	1	0	1
1	1	0	1	0	1	0	0	1	1
1	1	0	1	0	1	0	1	2	1
1	1	0	1	0	1	1	0	3	1
1	1	0	1	0	1	1	1	4	1
1	1	0	1	1	0	0	0	5	1
1	1	0	1	1	0	0	1	6	1
1	1	0	1	1	0	1	0	7	1
1	1	0	1	1	0	1	1	8	1
1	1	0	1	1	1	0	0	9	1
1	1	0	1	1	1	0	1	A	1
1	1	0	1	1	1	1	0	B	1
1	1	0	1	1	1	1	1	C	1
1	1	1	0	0	0	0	0	E	0
1	1	1	0	0	0	0	1	F	0
1	1	1	0	0	0	1	0	0	1
1	1	1	0	0	0	1	1	1	1
1	1	1	0	0	1	0	0	2	1
1	1	1	0	0	1	0	1	3	1
1	1	1	0	0	1	1	0	4	1
1	1	1	0	0	1	1	1	5	1
1	1	1	0	1	0	0	0	6	1
1	1	1	0	1	0	0	1	7	1
1	1	1	0	1	0	1	0	8	1
1	1	1	0	1	0	1	1	9	1
1	1	1	0	1	1	0	0	A	1
1	1	1	0	1	1	0	1	B	1
1	1	1	0	1	1	1	0	C	1
1	1	1	0	1	1	1	1	D	1

表 12　4 ビット加算器の設計(7/7)

A				B				結果	
SW9	SW8	SW7	SW6	SW5	SW4	SW3	SW2	LED	C
1	1	1	1	0	0	0	0	F	0
1	1	1	1	0	0	0	1	0	1
1	1	1	1	0	0	1	0	1	1
1	1	1	1	0	0	1	1	2	1
1	1	1	1	0	1	0	0	3	1
1	1	1	1	0	1	0	1	4	1
1	1	1	1	0	1	1	0	5	1
1	1	1	1	0	1	1	1	6	1
1	1	1	1	1	0	0	0	7	1
1	1	1	1	1	0	0	1	8	1
1	1	1	1	1	0	1	0	9	1
1	1	1	1	1	0	1	1	A	1
1	1	1	1	1	1	0	0	B	1
1	1	1	1	1	1	0	1	C	1
1	1	1	1	1	1	1	0	D	1
1	1	1	1	1	1	1	1	E	1

4 ビット加算器の加算が正しく得られていることがわかる。

6.2.3　ALU の設計

ALU の実験では、A の値と B の値を変え、S_1、S_0、C_0 の値を変えて真理値表を作成した。

(1)　A を 1111、B を 0000 とした場合

A を 1111、B を 0000 とした時の真理値表を表 13 に示す。

表 13　A を 1111、B を 0000 とした場合の真理値表

C0	S1	S0	LED	C	Y
0	0	0	F	0	0000
0	0	1	F	0	0000
0	1	0	E	1	1111
0	1	1	E	1	1111
1	0	0	0	1	0000
1	0	1	0	1	0000
1	1	0	F	1	1111
1	1	1	F	1	1111

表 2 の真理値表を用いて比較を行うと、正しい結果が得られていると考えることができる。

(2) A を 0000、B を 1111 とした場合

A を 0000、B を 1111 とした時の真理値表を表 14 に示す。

表 14　A を 0000、B を 1111 とした場合の真理値表

C0	S1	S0	LED	C	Y
0	0	0	0	0	0000
0	0	1	F	0	1111
0	1	0	0	0	0000
0	1	1	F	0	1111
1	0	0	1	0	0000
1	0	1	0	1	1111
1	1	0	1	0	0000
1	1	1	0	1	1111

(3) A を 0111、B を 0001 とした場合

A を 0111、B を 0001 とした時の真理値表を表 15 に示す。

表 15　A を 0111、B を 0001 とした場合の真理値表

C0	S1	S0	LED	C	Y
0	0	0	0	0	0000
0	0	1	8	1	0001
0	1	0	5	1	1110
0	1	1	6	1	1111
1	0	0	8	0	0000
1	0	1	9	1	0001
1	1	0	6	1	1110
1	1	1	7	1	1111

6.3　カウンタ回路の設計

6.3.1　非同期式カウンタ回路の設計

非同期式カウンタのロジックアナライザの出力波形を図 16 に示す

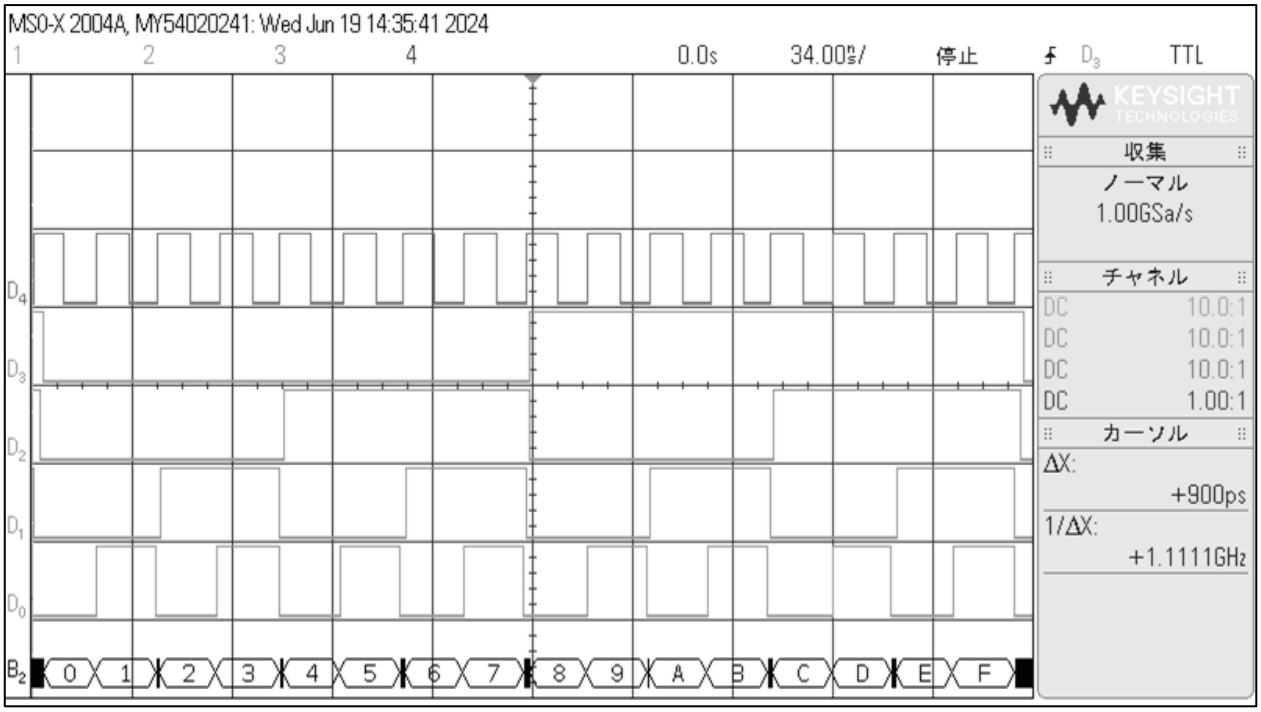

図 16　非同期式カウンタの出力波形

非同期式カウンタの波形より、すべての出力が立ち下がる瞬間を探し、各出力間の立ち下り時間幅を測定した。結果を表 16 に示す。

表 16　非同期式カウンタの遅延時間の測定結果

	遅延時間[ns]										
	1回	2回	3回	4回	5回	6回	7回	8回	9回	10回	平均
D0-D1	1	1	0.1	1	1	0.1	1	0	0	1	0.62
D1-D2	2	2	2	1.9	2	2	1.9	2	2.1	1.9	1.98
D2-D3	0.9	0.9	1	0.1	0.9	1	0.1	0.9	1	0	0.68
合計	3.9	3.9	3.1	3	3.9	3.1	3	2.9	3.1	2.9	3.28

結果として、最初の D0-D1 間の平均遅延時間は 0.62[ns]で、D1-D2 間は 1.98[ns]、D2-D3 間は 0.68[ns]、合計遅延時間の平均は 3.28[ns]となった。

6.3.2　同期式カウンタ回路の設計

同期式カウンタのロジックアナライザの出力波形を図 17 に示す

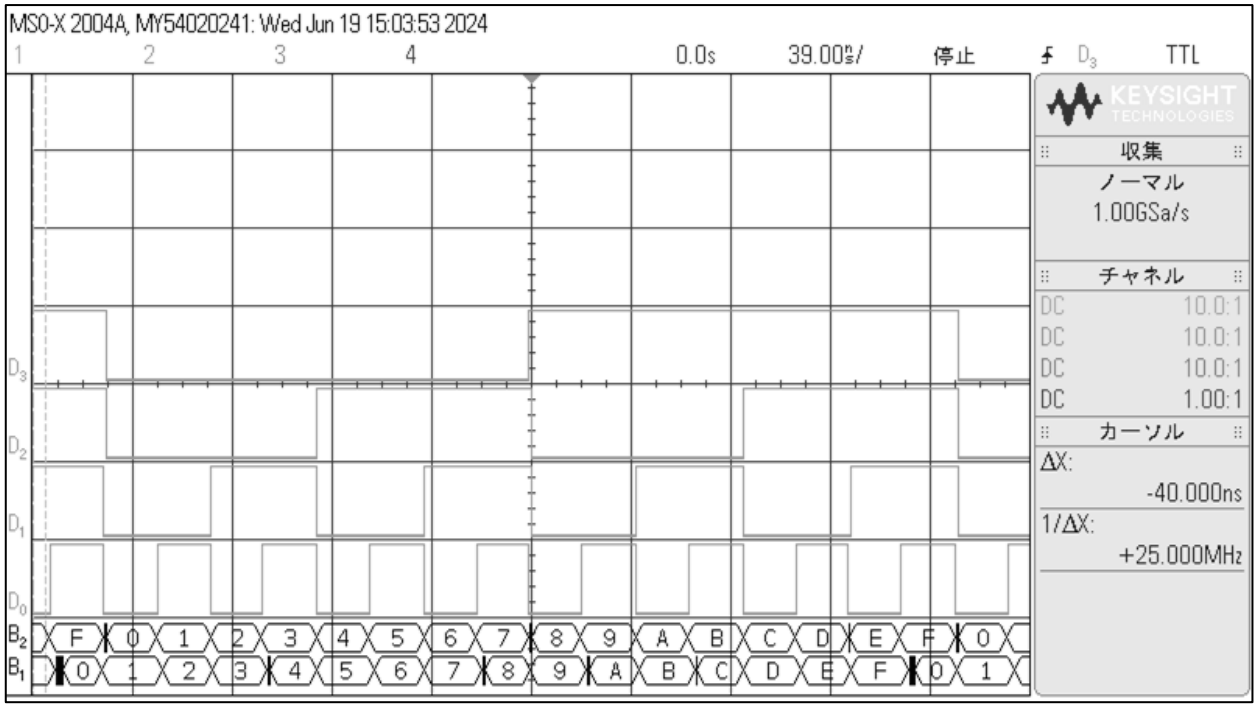

図 17　同期式カウンタの出力波形

同期式カウンタの波形より、すべての出力が立ち下がる瞬間を探し、各出力間の立ち下り時間幅を測定した。結果を表 17 に示す。

表 17　同期式カウンタ

	遅延時間[ns]										
	1回	2回	3回	4回	5回	6回	7回	8回	9回	10回	平均
D0-D1	0.1	1	1	0	1	1	0	1	1	0	0.61
D1-D2	1	0	0	0	0	0	0	0	0	1	0.2
D2-D3	0	0	0	1	0	0	1	0	0	0	0.2
合計	1.1	1	1	1	1	1	1	1	1	1	1.01

結果として、最初の D0-D1 間の平均遅延時間は 0.61[ns]で、D1-D2 間は 0.2[ns]、D2-D3 間は 0.2[ns]、合計遅延時間の平均は 1.01[ns]となった。

6.4　PWM 発生回路の設計

PWM 発生回路の測定ではこちらで Duty 比を変更しながら作成を行った。デジタルコードを 2 進数で 0000-1111 までをそれぞれ確認した。出力波形をそれぞれ 0000 から 1111 までを図 18 から図 33 に示す

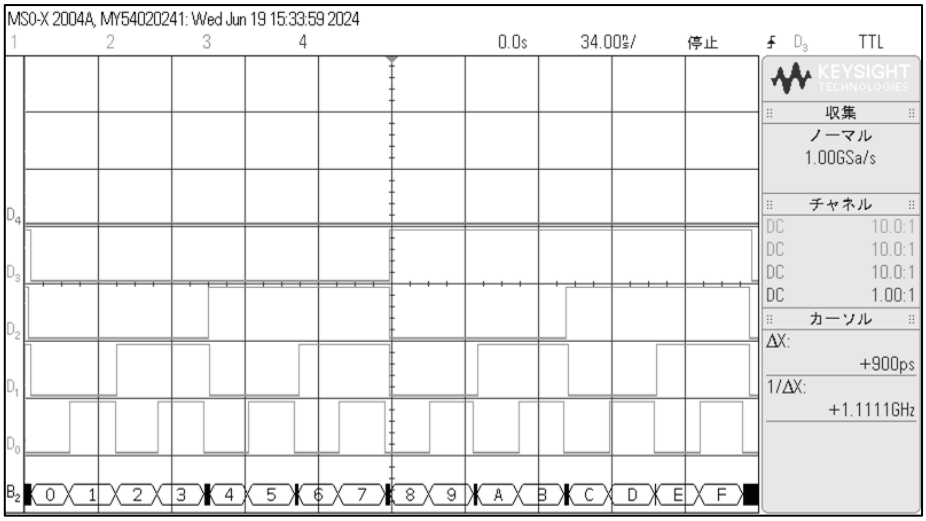

図 18　PWM 発生回路の出力波形（デジタルコード：0000）

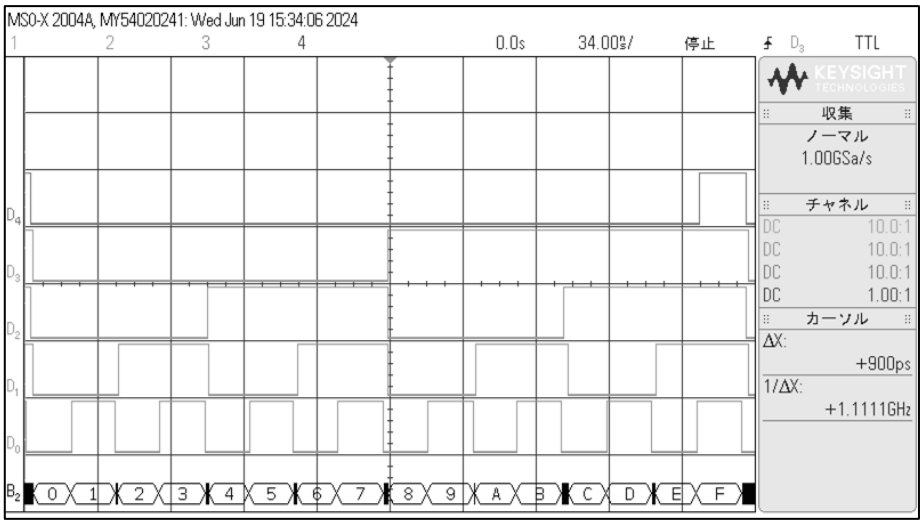

図 19　PWM 発生回路の出力波形（デジタルコード：0001）

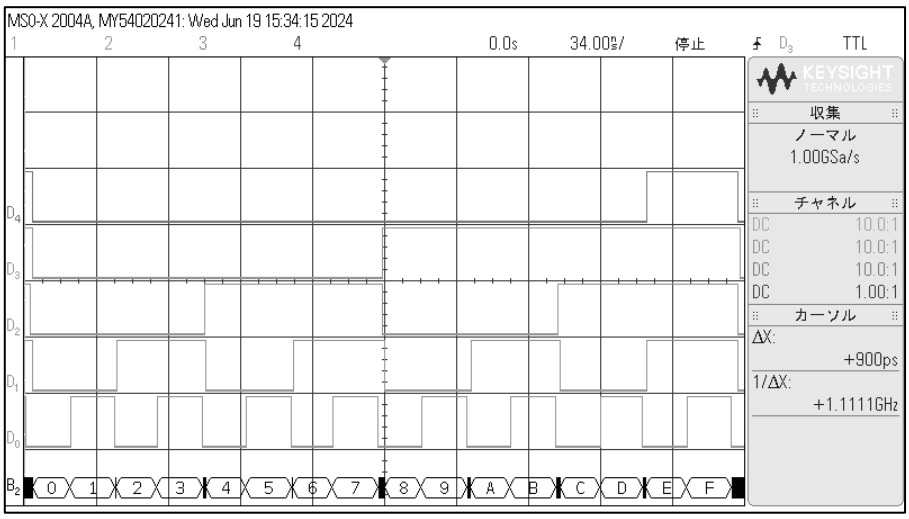

図 20　PWM 発生回路の出力波形（デジタルコード：0010）

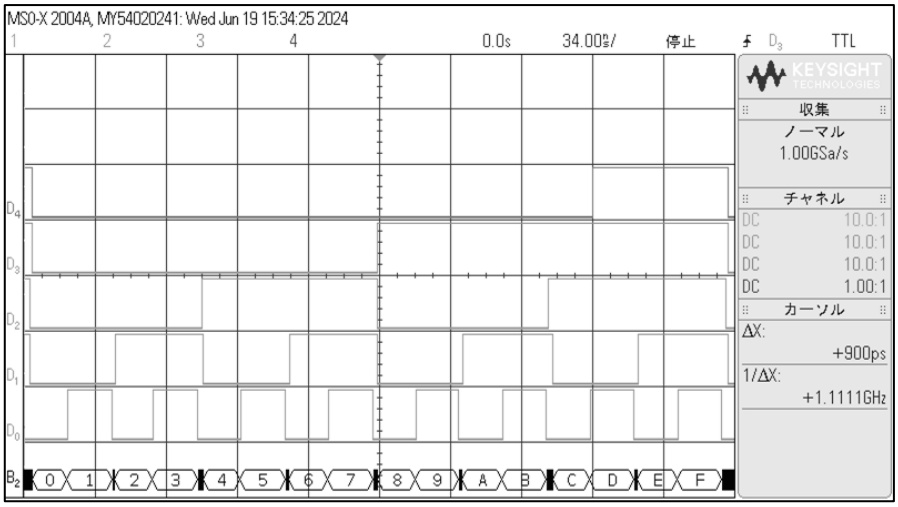

図 21　PWM 発生回路の出力波形（デジタルコード：0011）

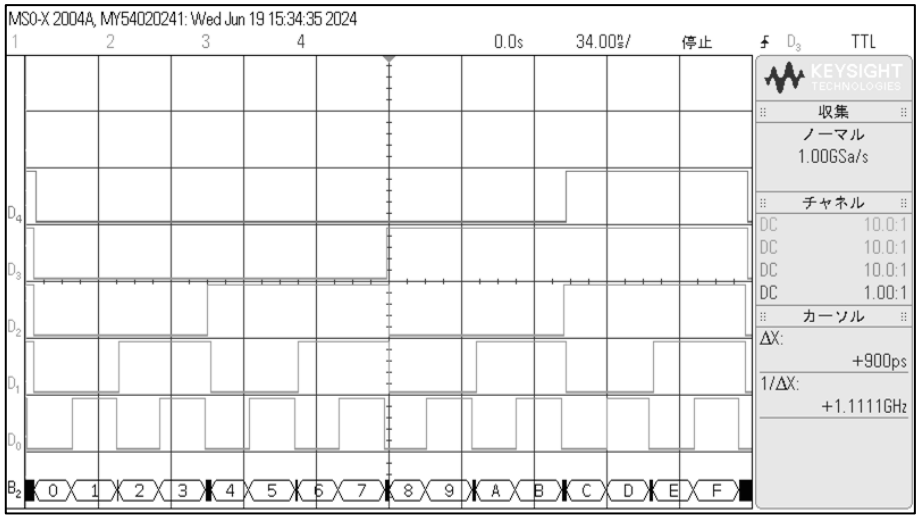

図 22　PWM 発生回路の出力波形（デジタルコード：0100）

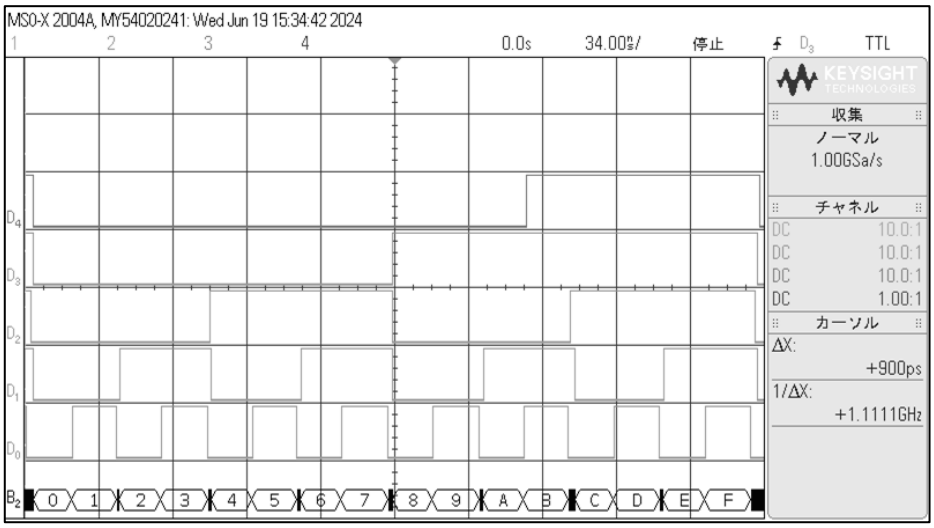

図 23　PWM 発生回路の出力波形（デジタルコード：0101）

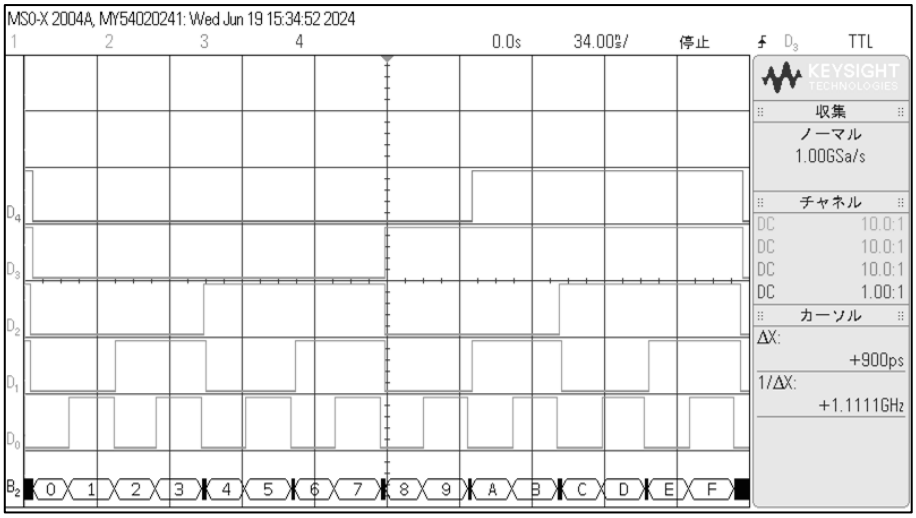

図 24　PWM 発生回路の出力波形（デジタルコード：0110）

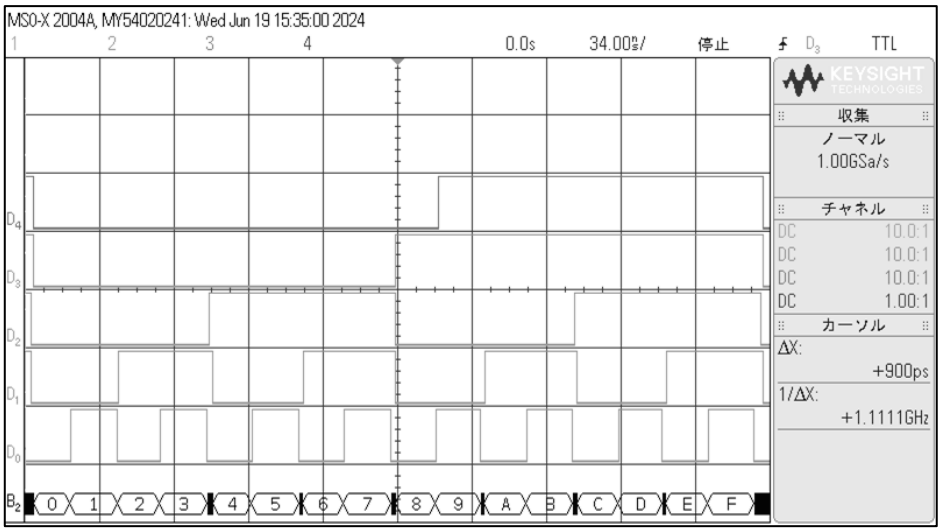

図 25　PWM 発生回路の出力波形（デジタルコード：0111）

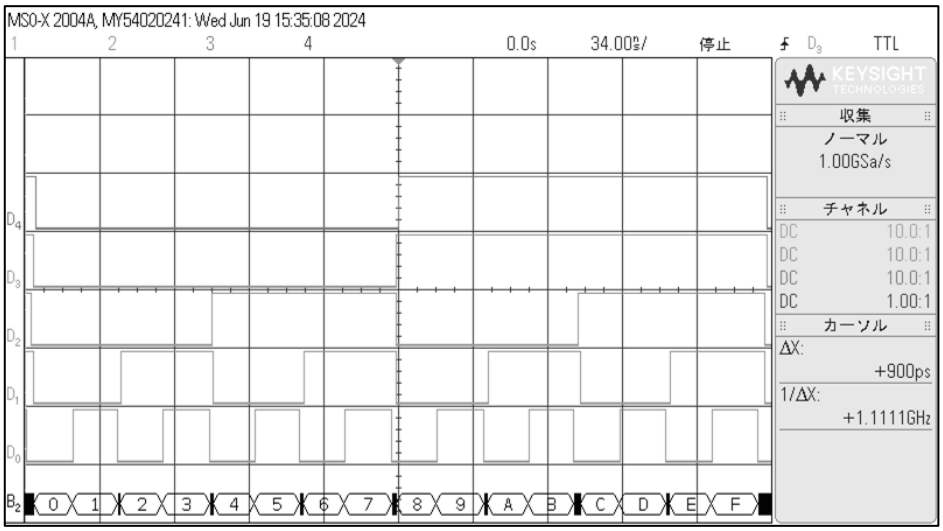

図 26　PWM 発生回路の出力波形（デジタルコード：1000）

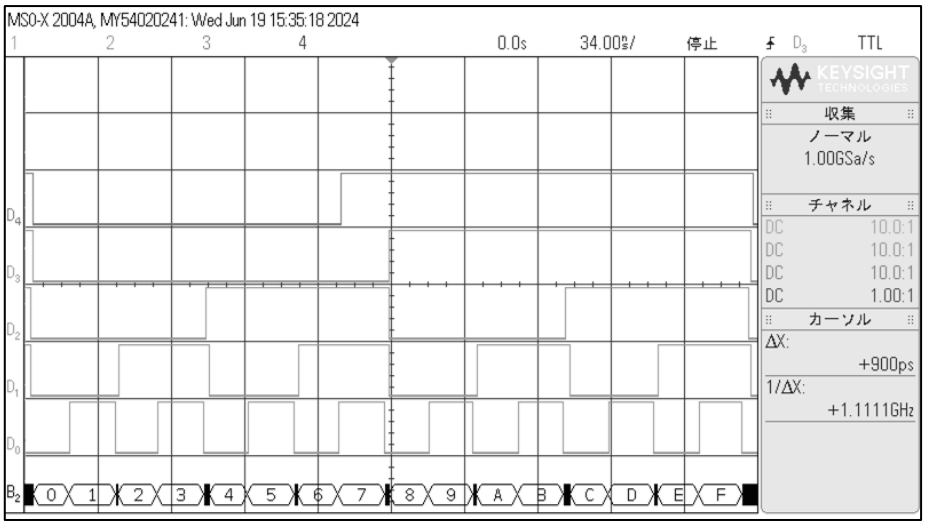

図 27　PWM 発生回路の出力波形（デジタルコード：1001）

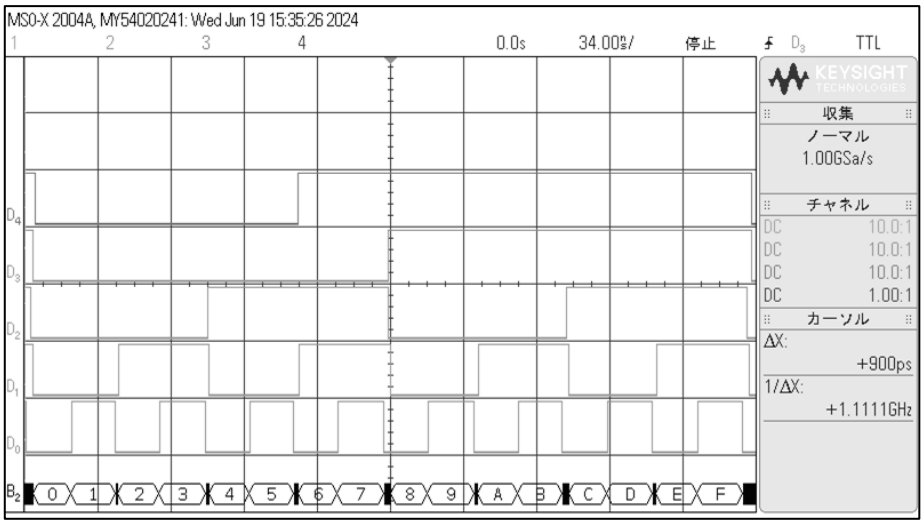

図 28　PWM 発生回路の出力波形（デジタルコード：1010）

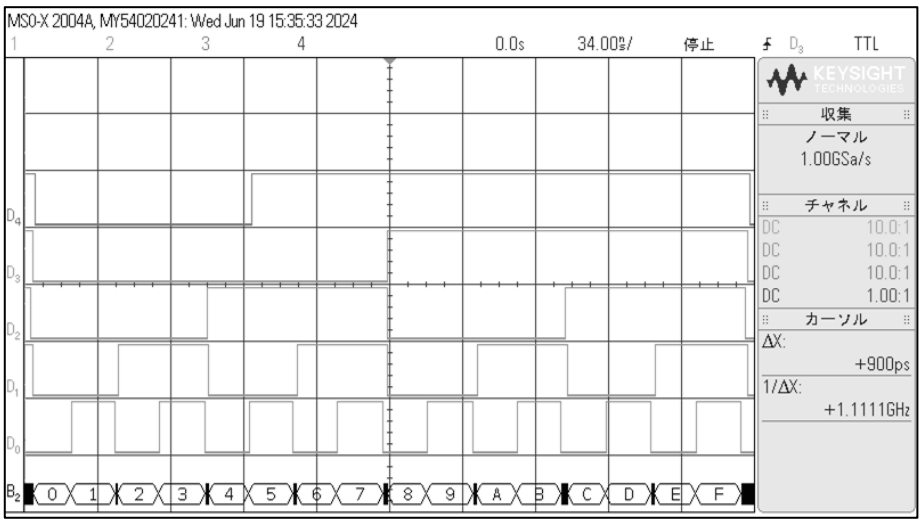

図 29　PWM 発生回路の出力波形（デジタルコード：1011）

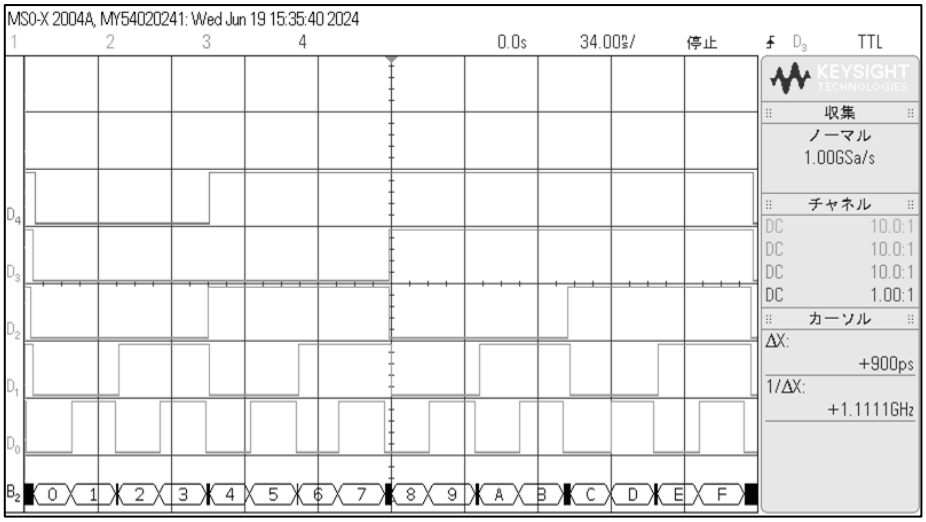

図 30　PWM 発生回路の出力波形（デジタルコード：1100）

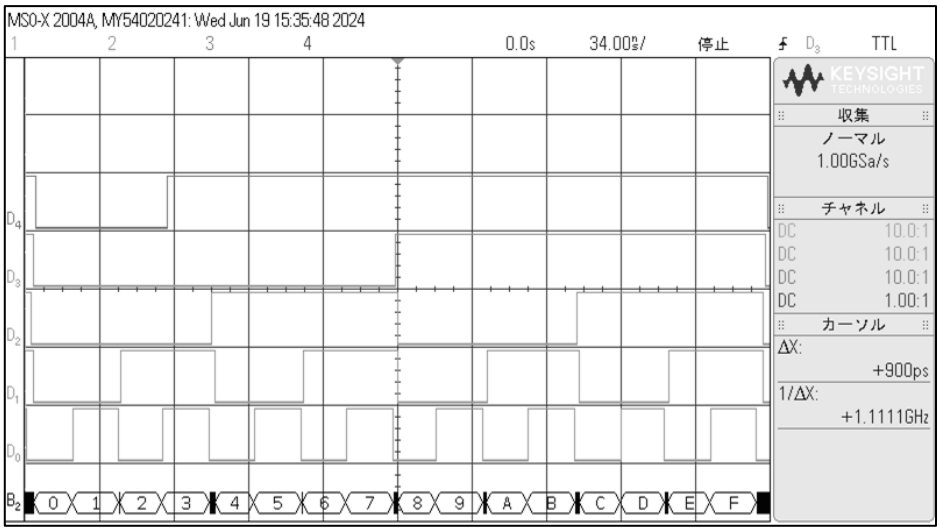

図 31　PWM 発生回路の出力波形（デジタルコード：1101）

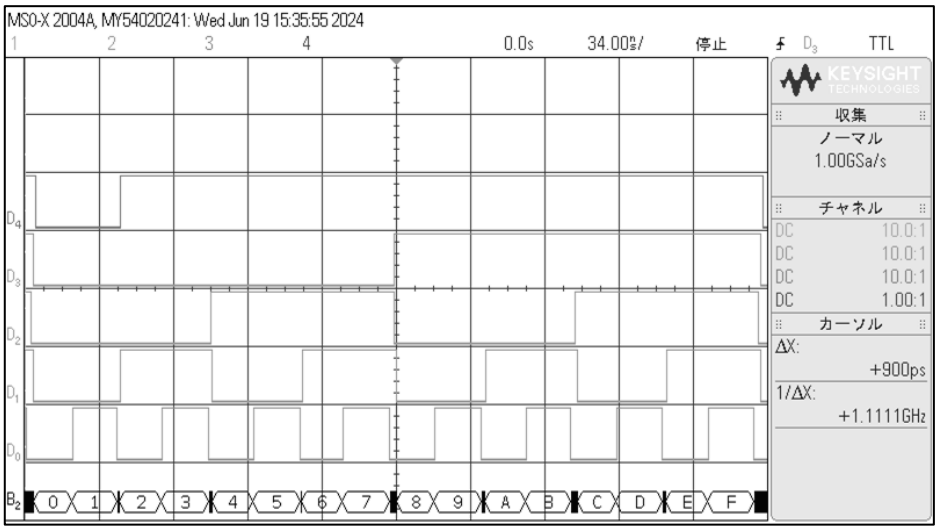

図 32　PWM 発生回路の出力波形（デジタルコード：1110）

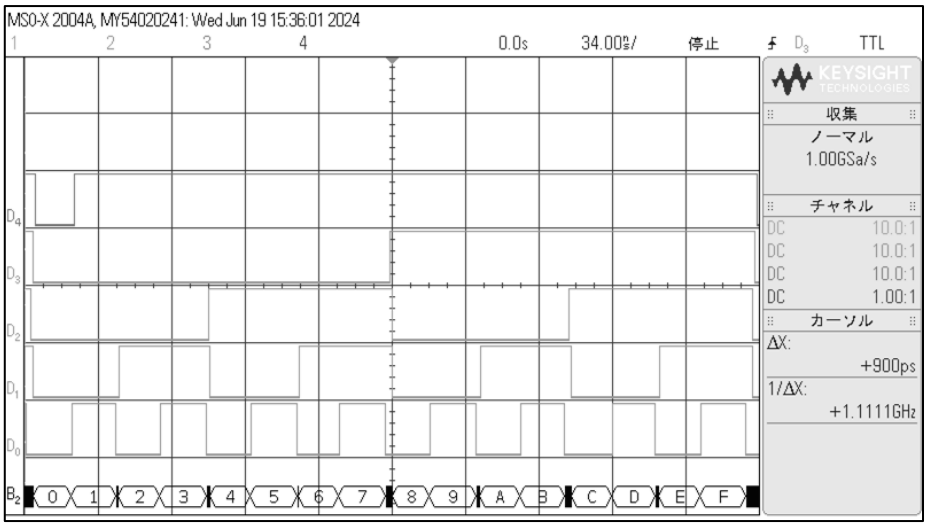

図 33　PWM 発生回路の出力波形（デジタルコード：1111）

図をそれぞれ比較すると、D4 の波形がそれぞれデジタルコードを増やしていくたびに増加していることがわかる。また、D4 の波形は PWM の出力を表しているため、デジタルコード、つまり Duty 比に比例して出力が増加していることがわかる。

次に、デジタルコードを増加させたときの PWM の出力電圧を表 18 に示す。

表 18　デジタルコードを変化させた時の出力電圧

SW9	SW8	SW7	SW6	Duty比	出力電圧[V]
0	0	0	0	0	0.0012
0	0	0	1	1/16	0.2124
0	0	1	0	2/16	0.4181
0	0	1	1	3/16	0.6299
0	1	0	0	4/16	0.8285
0	1	0	1	5/16	1.0404
0	1	1	0	6/16	1.2476
0	1	1	1	7/16	1.4602
1	0	0	0	8/16	1.6553
1	0	0	1	9/16	1.8657
1	0	1	0	10/16	2.0753
1	0	1	1	11/16	2.2856
1	1	0	0	12/16	2.4885
1	1	0	1	13/16	2.6961
1	1	1	0	14/16	2.9037
1	1	1	1	15/16	3.1151

また、横軸にデジタルコード、縦軸に PWM 出力電圧をとったグラフを図 34 に示す。

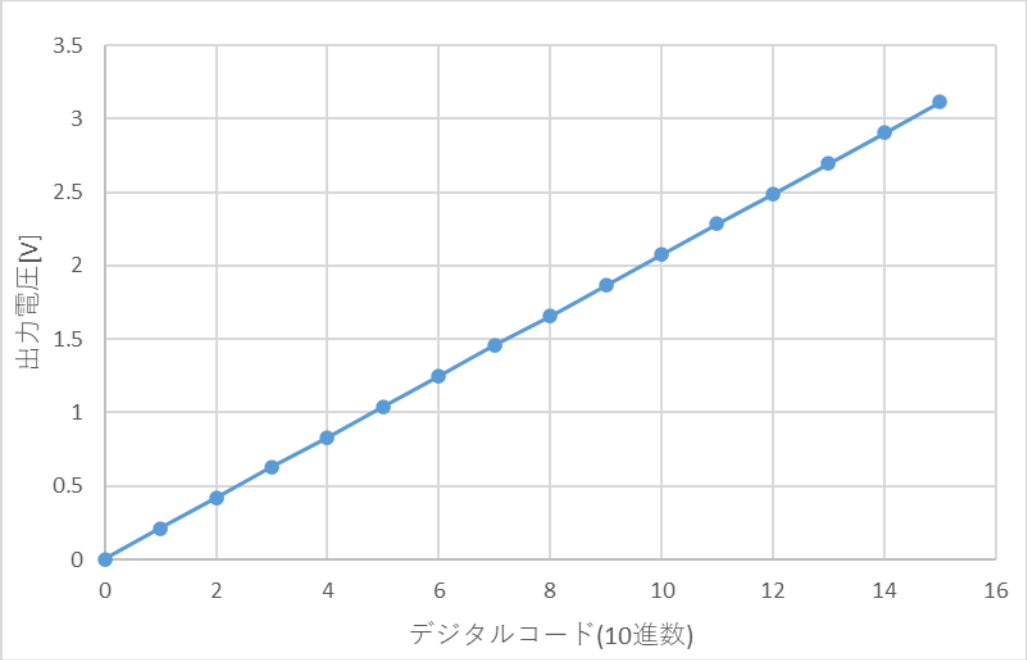

図 34　PWM 出力電圧の変化グラフ

グラフおよび数値から確認すると、値が Duty 比の値に比例して増加していることがわかる。

7. 考察

7.1　考察課題

7.1.1　組み合わせ回路の設計について、FPGA による測定結果がシミュレーション結果と一致していることを確認することとともに、簡単化の有無によるシミュレーション結果の差異について検討する。

FPGA の測定結果については表 4 の結果より分かる。表 6 では、入力が 010、011、111 の時に出力が 1 を示している。その結果をもとに、論理式を作成し、シミュレーションを行っている。実際のシミュレーション結果が図 11 である。/logic/output という箇所があり、そこで波形が 1 を示している箇所の上にある入力の値を確認すると、010、011、111 の時に 1 になっていることがわかる。そのため、シミュレーション結果と一致していることがわかる。また、シミュレーション結果の/logic/temp(i)では、(1)式の AND 部分の結果を確認することができる。

次に、簡単化を行った出力波形図 15 と簡単化を行う前の図 11 の比較を行うと、結果が一致していることがわかる。しかし、式が異なるため、それぞれの AND 回路部分を示す/logic/temp(i)部分は異なっている。また、簡単化を行ったことで、AND 部分が一つ少なくなったため、/logic/temp(2)が常に 0 を示すようになっている。

論理式の簡単化を行う理由については、式の簡略化の他にも、現実だと大きな利点が存在する。現実では回路素子を増やせば増やすほど、値段がかかってしまう。そのため、簡単化を行うことで回路素子の量を減らし、費用の削減が行える。また、産業化すると、回路を量産する必要があるため、簡単化を行うことで、一個一個が微小でもかなりの費用を削減できる。

7.1.2　演算回路の設計において、ALU 回路の出力結果が持つ演算の意味について、実際のコンピュータで使用することを想定したうえで検討する。

演算回路の設計にて設計した ALU 回路は図 5 の回路図と同じ動作をするようになっているため、図を理解することで説明ができる。図 5 の ALU 回路では、事前に選択信号を変更することができる。選択信号を変えると何が起こるのかだが、今回の ALU 回路だと B の値をどのように処理するのかと、値に対して+1 を行うかなどを変えることができる。具体的に確認すると、表 2 に示すような真理値表となることがわかる。表 2 の真理値表では、選択信号が関係していない X の箇所では常に A の値が出力されることになっているが、Y の箇所では選択信号、S_0 と S_1 を変更したときに Y の箇所が変わっていることがわかる。具体的には、S_0 を 1 にした場合には B の値を出力しており、S_1 の場合は B の否定を出力している。どちらも 1 だった場合は 1 を出力していることが図からわかる。すべての選択信号を 1 にした場合の回路は A の値+桁上げ 1 を出力する回路になることがわかる。

ALU 回路は、入力される値に対して、選択信号を変えることによって行う演算を変えることができる。選択信号が影響を及ぼす回路部分をセレクタ回路と呼ばれる。また、実際のコンピュータでは予習課題(2)で調べた通りに、CPU などの複数の演算を行わなければならないものに用いられており、実際にはさらに複雑な条件のもと運用されていると考えられる。

今回予習課題(5)で作成した真理値表を元に、実際に計算を行い、実験結果と比較を行った。実験結果は A が 7(0111)、B が 1(0001)のものを用いて計算を行った。

表 19　ALU 回路の実験結果の検算

C0	S1	S0	X	Y	Z	実験結果
0	0	0	0111	0	7(0111)	0(0000)
0	0	1	0111	0001	8(1000)	8(1000)
0	1	0	0111	1110	5(0101)	5(0101)
0	1	1	0111	1111	6(0110)	6(0110)
1	0	0	0111	0	8(1000)	8(1000)
1	0	1	0111	0001	9(1001)	9(1001)
1	1	0	0111	1110	6(0110)	6(0110)
1	1	1	0111	1111	7(0111)	7(0111)

表 19 を確認すると、C0、S1、S0 がともに 0 の場合のみ異なっていたが、それ以外はしっかりと実験結果と同じ数値を示していることが分かった。異なっていた理由としては、LED の観測を正しく行っていなかったからだと考えられる。実際に、他の実験の A が 15(1111)の場合や、A が 0(0000)の場合では正しい値を確認できているため、この実験では測定ミスが起きていることが分かった。

7.1.3　カウンタ回路の設計において、同期式と非同期式における出力波形や遅延時間の差異について検討する。

カウンタ回路は図 1 のような非同期式カウンタ回路と、図 2 のような同期式カウンタ回路がある。今回それぞれの回路において、出力波形および平均遅延時間を計測した。

非同期式カウンタ回路の出力波形の図 16 では、D4 のクロックで、D3 から D0 のそれぞれが D-FF

の出力で、B2 がカウンタの出力を表している。遅延時間は表 16 に示していて、D-FF のそれぞれの遅延時間の平均の合計では 3.28[ns]遅延していることが分かった。

同期式カウンタの出力波形の図 17 では、クロックが省かれており、その代わりに B2 で最終的なカウンタの出力と、B1 に途中経過のカウンタ出力を示している。遅延時間は表 17 に示していて、D-FF のそれぞれの遅延時間の平均の合計では 1.01[ns]遅延していることが分かった。

非同期式カウンタ回路にて遅延時間が生じる原因は図 35 に示すような動作をしているからである。

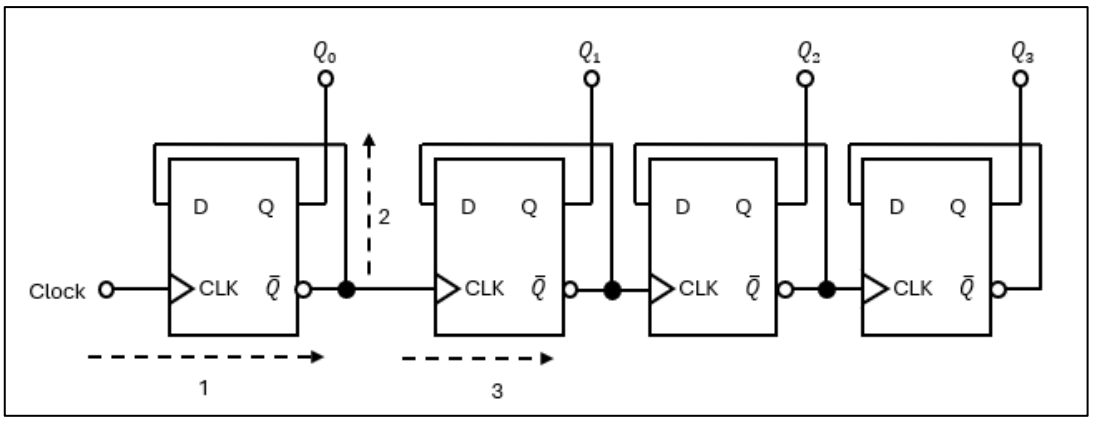

図 35　非同期式カウンタ回路の動作

非同期式カウンタでは、一度否定した出力をもう一度 D-FF に入力した後に次の D-FF に出力しているため、遅延時間が少しずつ蓄積してしまう。そのため、後半に行けば行くほど最初の出力から遅延が生じてしまっている。

次に、同期式カウンタで遅延時間が少ない理由は、図 36 のような動作をしているからである。

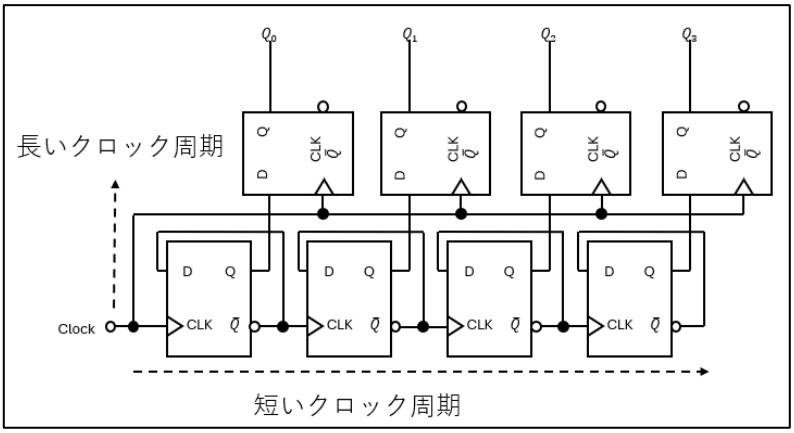

図 36　同期式カウンタ回路の動作

同期式カウンタ回路では二つの非同期式カウンタ回路を用意しており、一度短いクロック周期で非同期式カウンタ回路を動かし、その出力をもう一個の長いクロック周期を受け取った非同期式カウンタ回路に入力することで、遅延を少なくしている。実際に、B1 が短いクロック周期の非同期式カウンタ回路の出力であり、B2 が長いクロック周期を用いた非同期式カウンタ回路の出力である。

7.1.4　PWM 発生回路の設計において、その動作原理と出力波形との関連性を示すとともに、Duty 比を変化させたときの PWM 出力電圧特性について検討する。

PWM 発生回路は図 8 のように構成されている。しかし、同期式カウンタ回路の出力が図 8 では B に入力されているが、今回構成した回路では A に入力されているという点が異なっている。

今回作成した PWM 発生回路は、動作として、カウンタ回路の出力を A に入力し、Duty 比を B に入力している。つまり、カウンタ回路で 0 から 15 の出力を足して、そこに Duty 比を足すことによって桁上げを用意している。また、クロック周期であるため、Duty 比はどの程度桁上げするかを示しており、電圧の場合は桁上げのあった時間に応じて、電圧を出力する時間が変わる。PWM は、電圧を変えるために、パルス周期の平均が出力電圧になるという方法を使って出力電圧を変更している。そのため、桁上げの時間が増えるということは、パルス周期が変更されているということで、その影響で出力電圧が変わっている。

PWM 発生回路の出力波形では、D4 が PWM の出力を表しており、図 8 の最終的な桁上げを表している。そのため、桁上げが 1 の時に電圧を出力しており、0 の時は電圧を出力していない。次に、D3 から D0、および B2 でカウンタ回路の出力を表している。これによって、クロック周期に対応した桁上げが作成することができる。出力波形の図 18 を確認すると、Duty 比なし、つまり 0 入力の場合では PWM の桁上げが生まれないため、一度も出力が 1 になっていない。図 19 では Duty 比が 1/16 になっているため、最後に桁上げが発生している。最後の図 33 では 15/16 であるため、最初以外は常に 1 になっていることがわかる。

Duty 比を変化させることによって、出力が増えていることが波形でわかる。そのため、出力は Duty 比に比例している、出力時間が増えるということは、パルス周期が短くなっているため、出力電圧が比例して増えると考えることができる。実際に、テスタで電圧を確認した表 18 では、Duty 比を増やしていくたびに、電圧が上昇していることがわかる。図 34 のグラフでも完全に一直線になっているため、Duty 比に比例していることがわかる。

7.2　実験考察

7.2.1　同期式カウンタ回路と非同期式カウンタ回路の強みと弱み

非同期式カウンタ回路では、遅延時間が生じてしまうという弱みが存在する。しかし、その点を除けば、すぐカウントができるため、より正確な時間が必要でない限り、回路素子が少なく済むため、予算の削減ができるだろう。

同期式カウンタ回路では、遅延時間が少なくなるという強みが存在する。しかし、単純な計算で非同期式カウンタ回路の 2 倍の回路素子が必要なため、かなり費用がかかってしまう。そのため、よほど遅延時間が影響を及ぼすようなものでない限り、非同期式カウンタ回路の方がいいのではないかと考えられる。

8. 参考文献

[1]　論理回路ノート、高橋　寛、コロナ社、2021 年 12 月

[2]　組込みシステム概論、戸川　望、CQ 出版社、2019 年 4 月