DA変換の原理 レポート　3I44 吉髙 僚眞

目的

アナログ電圧をコンピュータで扱う場合はディジタル値に変換する。この処理を A/D 変換という。分解能 10 ビットの場合、基準電圧を５V とすると、5/210 V 毎（1024 段階）に量子化されることになる。ここでは４ビット逐次比較型を扱う。また同時に、データシートを用いたプログラミングについて学ぶ.

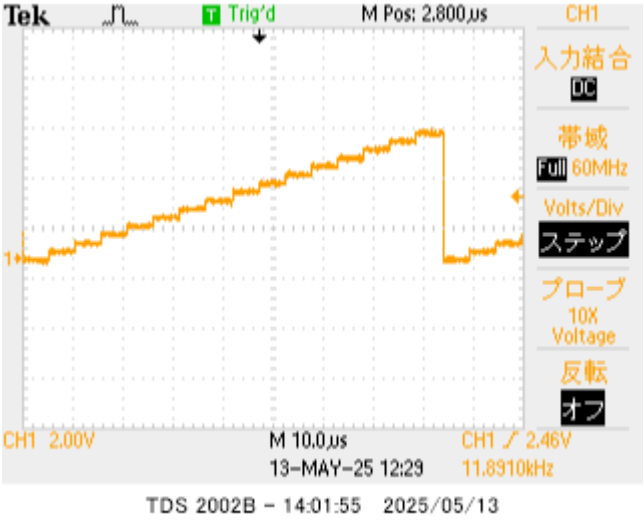
実験1. 抵抗ラダー型D/A

1. AVR に４ビット DA 変換回路を接続し、Vo に PB（PB3 PB2 PB1 PB0）にほぼ比例した電圧が出力されることを確かめよ。

   プログラム

```
#include <asf.h> //int.c
#define F_CPU 20000000UL
#include <util/delay.h>

void io_init(void);

int main (void)
{
    io_init(); //IO ポート設定
    while(1){
        for(int i=0; i<16; i++){
            PORTB = i;
            _delay_us(5);
        }
    }
    return 0;
}

void io_init(void) //IO ポート設定
{
    DDRB = 0b00001111;
    // +---------- PB0~3 を出力に設定
    return;
}
```

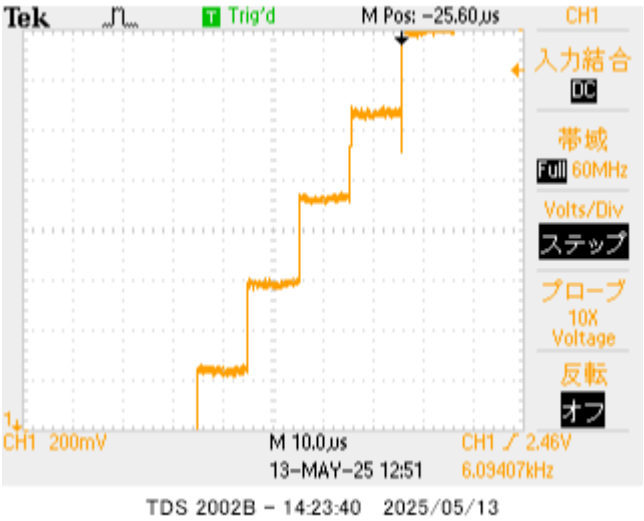
   波形



0b00000001~0b00001111まで順に5µ秒ごとに出力したところ、階段状の波形が見られたことから、ほぼ比例した電圧が出力されていることがわかる。

2. DA 変換回路の最小分解能（1 LSB）の電圧を測定せよ。

   波形



先ほどと同様のプログラムでオシロスコープの設定を変更したところ、このような図を得ることができた。この図から、各出力電圧の差が300mV~400mVほどになっていることがわかる。
計算してみると、

$$\frac{5}{2^4} = 0.3125$$

より 312.5mVとなり、おおよそ妥当な値だとわかる。

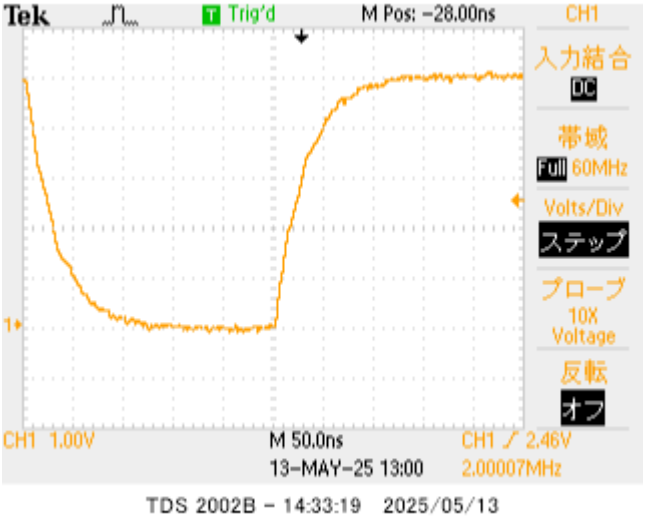
3. PB にディジタル値の最小値、最大値を交互に繰り返し与え、Vo の変化を観察せよ。また Vo が安定するまでの時間（遅延時間）を求めよ。安定するまでの時間とは電圧誤差が最小分解能（1 LSB）以内に収まるまでの

時間とせよ。

プログラム

```
// ・・・省略
int main (void)
{
    io_init(); //IO ポート設定
    PORTB = 0b00000000;
    while(1){
        PORTB ^= 0b00001111;
    }
    return 0;
}
// ・・・省略
```

波形



波形を見ると、0Vから5V近くになるまでおおよそ150n秒(3×50)ほどかかっていることがわかる。

実験2. コンパレータ

1. テキスト「コンパレータの使い方」をもとにコンパレータを設定し、動作を確認しなさい。V+に三角波（ファンクションジェネレータ）、V-に定電圧（３V）を与えオシロスコープで確認すること。

テキスト「コンパレータの使い方」より

3. 概要

> ① 比較器を使用するには、ACDを ON する。 (論理値0にする)
>
> ② 入力端子は＋端子のAIN0とー端子のAIN1であり、この電圧を比較する。
>
> ③ 比較するには、ACBGを論理値0、ACMEを論理値0またはADENを論理値1にすればよい。

> ④ 比較結果はACOに出力される。
>
> ⑤ この図の他、以下のページ、項目を参照する。
> 243 ページ Table 23-1
>
> 12 ページ Figure1-1
>
> 97 ページ Table 14-9

4. ピン配置

> ⑥ ピン配置は、図1-1 、表14-9を参照すればよい。
>
> ⑦ AIN0 はポートDの6ビット目、ICの12番ピンである。
>
> ⑧ AIN1 はポートDの7ビット目、ICの13番ピンである。

5.省電力設定

> ⑨ ADC については、246ページに書かれている。
>
> ⑩ ADC を使用するためには、PRADC を論理値0にする。
>
> PRADCの配置されているレジスタを探す
>
> ⑪ PRADCはPRRレジスタの0ビット目である。
>
> ⑫ PRADCが論理値1のとき、コンパレータは ADC input MUX を使用できない。
>
> ⑬ ディジタル入力にはディセーブル機能があり、DIDR1レジスタ（245ページ）、DIDR0レジスタ（260ページ）で設定する。
>
> ⑭ AIN1,AIN0を使用する場合、AIN1D, AIN0Dを論理値1にセットし、ディジタル入力バッファをディセーブルにすることができる。ADC5～0 も同様

6.そのほかのパラメータ

> ⑮ ACME はADCSRBレジスタの6ビット目である。
> 論理値0のとき、－入力端子 AIN1 がコンパレータに接続される。
>
> ⑯ ADEN はADCSRAレジスタの7ビット目である。
> 論理値0のとき、ADC が OFF になる。
>
> ⑰ ACO はACSRレジスタの5ビット目である。コンパレータ出力端子。
>
> ⑱ ACD はACSRレジスタの7ビット目である。
> 論理値1のとき、コンパレータが OFF になる。
>
> ⑲ ACBG はACSRレジスタの6ビット目である。
> 論理値0のとき、＋入力端子 AIN０がコンパレータに接続される。
>
> ⑳ ACD は論理値1で比較器の電源が OFF になる。
> 21 ACO には1~2クロック分のディレイが発生する。

7.まとめ

今回の目的で、０に設定するビット、１に設定するビットは次のようになる。

レジスタ	アドレス	0に設定するビット	1に設定するビット
PRR	0x64 PRADC	0	
ACSR	0x30(0x50) ACD, ACBG	7, 6	
ADCSRB	0x7B ACME	6	
ADCSRA	0x7A ADEN		7
DIDR0	0x7E ADC0~5		0,1,2,3,4,5
DIDR1	0x7F AIN0D, AIN1D		0, 1

8.遅延時間、タイミング

> 比較器の遅延時間(Propagation Delay)を調べる。表 30-1 の DC 特性を参照する。表から2.7Vで750ns（標準）、4Vで500ns（標準）である。遅延時間は一般に電圧が高いほど短くなるから、5Vでは480ns（最大）*と予想する。これは、20MHz動作で10クロック(0.48us)に相当する。
>
> さらに ACO がセットされるまでに2クロック必要だから、比較器への入力が安定後、合計12クロック（0.6us）必要であることがわかる。

9.プログラミング基礎知識（補足）

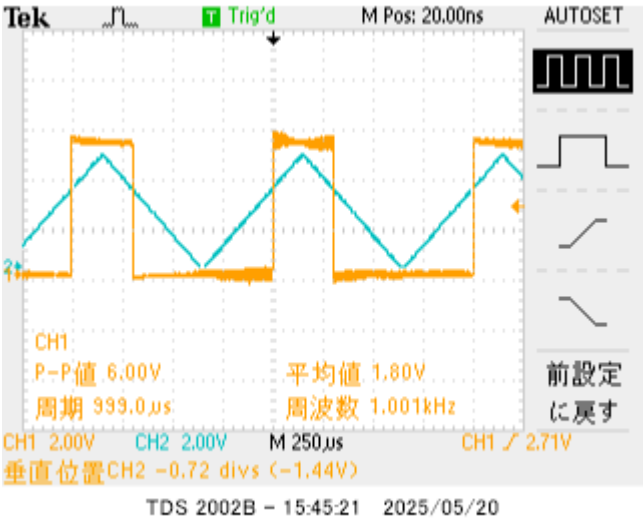
> 22 例えば、 PINB0x03番地、PORTB0x05番地、DDRB0x04番地 PRR0x64番地
>
> 23 IN, OUT命令は0x00 – 0x3F番地のアドレスに対して使用
>
> 24 LDS, STS命令は0x60 – 0xFF番地のアドレスに対して使用
>
> 25 SBI, CBI命令は0x00 – 0x1F番地のアドレスに対して使用

プログラム

```
#include <asf.h> //int.c
#define F_CPU 20000000UL
#include <util/delay.h>

void io_init(void);

int main (void)
{
    io_init(); //IO ポート設定
    PORTB = 0b00000000;
    while(1){
        if( (ACSR & (1<<5)) !=0){
            PORTB = 0b00010000;
        }else{
```

```
                PORTB = 0b00000000;
            }
            _delay_us(0.6);
        }
        return 0;
    }

    void io_init(void) //IO ポート設定
    {
        PRR    = PRR & (~(1<<0));
        ACSR   = ACSR & (~(1<<6));
        ACSR   = ACSR & (~(1<<7));
        ADCSRB = ADCSRB & (~(1<<6));
        ADCSRA = ADCSRA | (1<<7);
        DIDR0  = DIDR0 | (1<<0);
        DIDR0  = DIDR0 | (1<<1);
        DIDR0  = DIDR0 | (1<<2);
        DIDR0  = DIDR0 | (1<<3);
        DIDR0  = DIDR0 | (1<<4);
        DIDR0  = DIDR0 | (1<<5);
        DIDR1  = DIDR1 | (1<<0);
        DIDR1  = DIDR1 | (1<<1);

        DDRB   = 0b00010000;
        return;
    }
```

波形



実験3. A/D変換

1. アセンブリ言語を用いてプログラムを作成せよ。

プログラム

```
INIT:
    LDS R18, PRR
    ANDI R18, (~(1 << 0))
    STS PRR, R18

    LDS R18, ACSR
    ANDI R18, (~(1<<6))
    ANDI R18, (~(1<<7))
    STS ACSR, R18

    LDS R18, ADCSRB
    ANDI R18, (~(1<<6))
    STS ADCSRB, R18

    LDS R18, ADCSRA
    ORI R18, (1 << 7)
    STS ADCSRA, R18

    LDS R18, DIDR0
    ORI R18, (1 << 0)
    ORI R18, (1 << 1)
    ORI R18, (1 << 2)
    ORI R18, (1 << 3)
    ORI R18, (1 << 4)
    ORI R18, (1 << 5)
    STS DIDR0, R18

    LDS R18, DIDR1
    ORI R18, (1 << 0)
    ORI R18, (1 << 1)
    STS DIDR1, R18

    LDI R18, 0b00111111
    OUT DDRB, R18

START:
    ;全体の回数を保存するレジスタ
    LDI R20, 4
    ;ビットの桁数を保存するレジスタ
    LDI R21, 0b00010000
    ;出力する状態を保存するレジスタ
    LDI R22, 0b00000000
    OUT PORTB, R22
    RCALL DELAY_15CLOCK
LOOP:
    LSR R21
    OR R22, R21

    ; 1回目 0b00001000
    ; 2回目 0b00000100

    OUT PORTB, R22
    ;待ち処理 3回分 50ns * 3 = 150ns
```

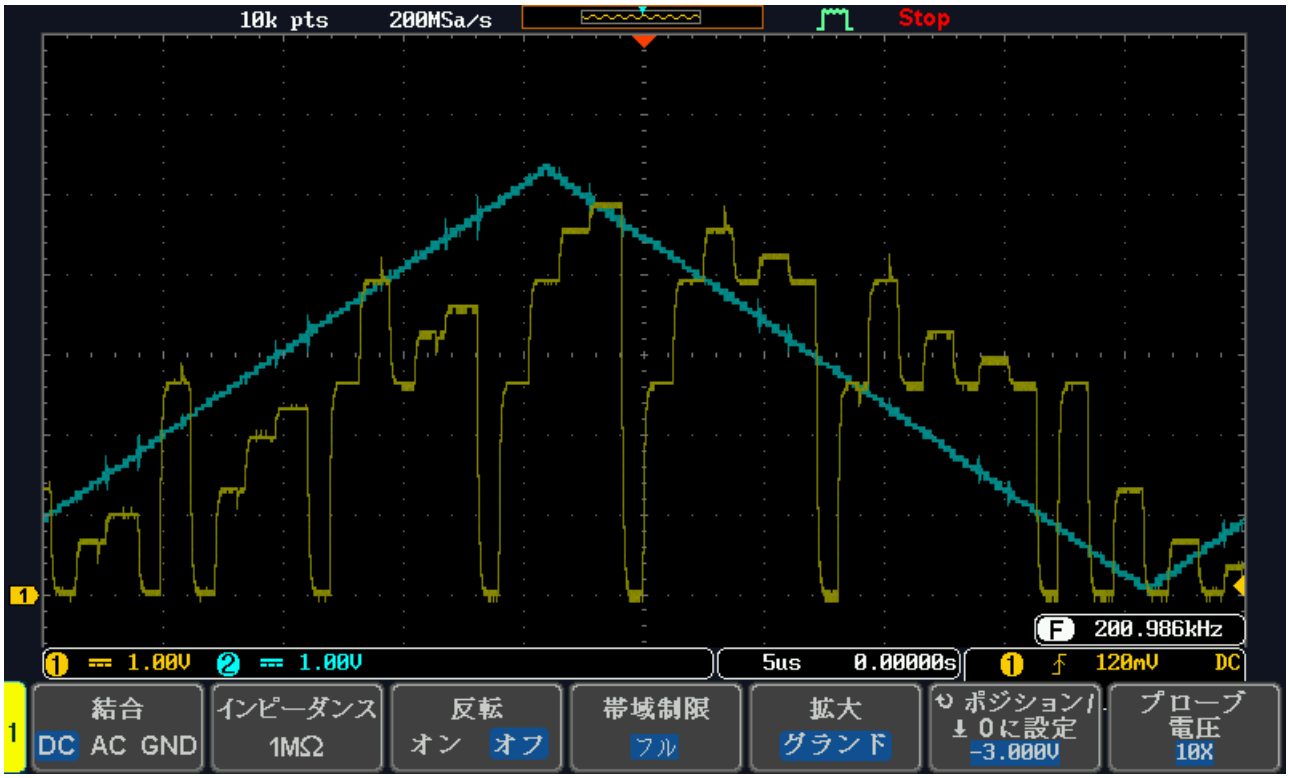
```
    RCALL DELAY_15CLOCK


    IN R18, ACSR
    ANDI R18, (1 << 5)

    BRNE DOWN
    SUBI R20, 1
    BRNE LOOP
    RJMP START
DOWN:
    MOV R23, R21
    COM R23
    AND R22, R23

    ;例: 2回目の処理
    ; R22 0b000001100;
    ; R23 0b111111011;
    ;     0b000001000;
    ;待ち処理

    ORI R22, (1 << 5)
    OUT PORTB, R22

    RCALL DELAY_15CLOCK

    SUBI R20, 1
    BRNE LOOP
    RJMP START

DELAY_15CLOCK:
    NOP
    NOP
    NOP
    NOP
    NOP
    NOP
    NOP
    NOP
    RET ; +4
    ;RCALLは3クロック
```

2. 電源から定電圧を入力し、ディジタル値が求められる様子を確認せよ。

   波形

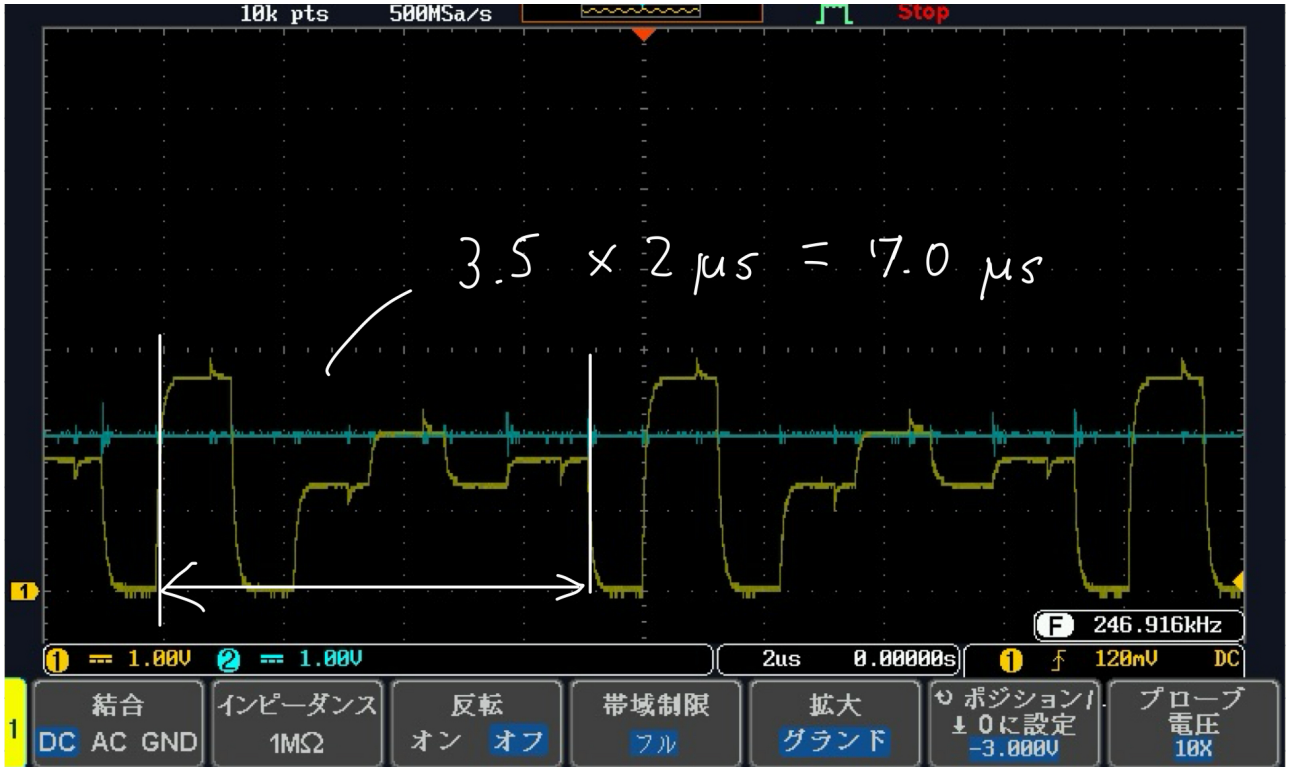
3. A/D 変換中に入力電圧が変化するとどのようになるか。実験で確認せよ。入力信号として SIN 波または三角波（ファンクションジェネレータ）を用いるとよい。

   波形



考察

1. A/D 変換に必要な時間はどのように考えたらよいか。実験結果に図示して説明せよ。



波形の図中からD/A変換で安定した電圧が出るまでに5μsのレンジで、3.5マス分必要であることがわかる。そのため、およそ`3.5 × 5μs = 7μs`ほど必要であると考えられる。

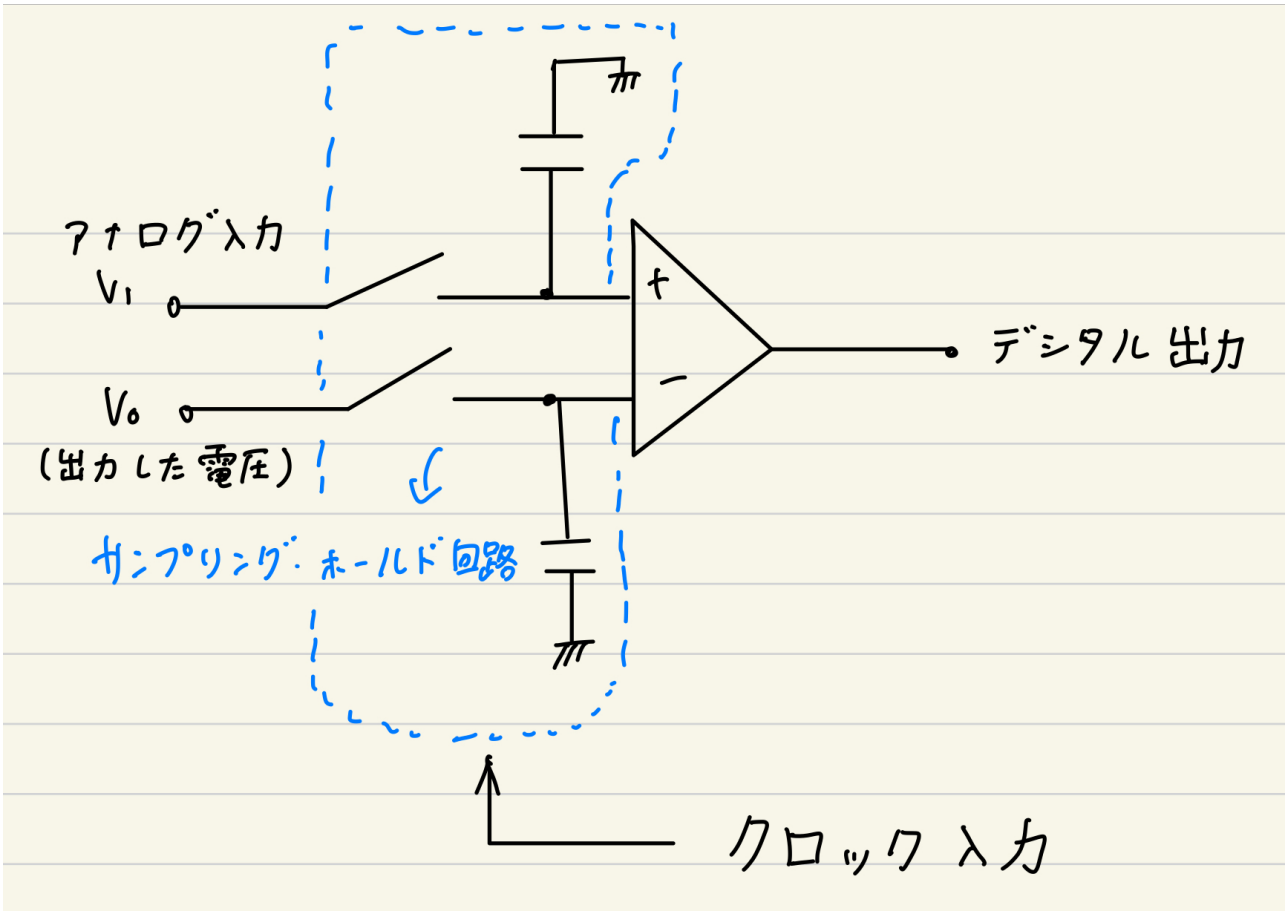
2. 今回のようなプログラムを作成する場合、C 言語、アセンブリ言語のどちらを使用するべきか。自分の考えを述べなさい。（利点、欠点を挙げ、重視すべき事項を比較）

C言語の場合、アセンブリ言語と比較して、プログラムが書きやすくデバックもしやすい。アセンブリ言語の場合、プログラムやデバックはC言語と比べてやりにくいが、C言語のコンパイル時の最適化などを考えずにプログラムを書くことができる。

また、1 命令＝何サイクルというところまで細かく挙動を調整できるため、C言語よりも時間的な制約などがより厳しい場合はアセンブリ言語を使う方が良いと思う。

この逐次D/A変換のプログラムの場合は、一定時間でD/A変換をする必要があるならばアセンブリで書いた方が良いと思う。そうでなく、とりあえず動かす必要があるならばC言語で描いた方が良いと思う。

3. A/D 変換器の前段にサンプリング・ホールド回路を用いることがある。これについて調べなさい。



サンプリング・ホールド回路を使うことで、A/D変換を行う間、アナログ信号をある一定の値として保持することができる。

クロック入力がHIGHTの時、サンプリング・スイッチがONとなり、出力にはアナログ入力がそのまま現れる。これをサンプリング・モードという。

入力がLOWの時、サンプリング・スイッチがOFFとなり、アナログ入力とデジタル出力が切り離される。この時、コンデンサによって保持された切り替わる前のアナログ入力が現れる。これをホールド・モードという。

この2つの状態によってA/D変換時にアナログ信号が変化してしまうことを防ぐことができる。

参考 高速A-D変換のしくみとIC活用術

4. A/D 変換には他にどんな方式にあるだろうか。調べなさい。

    - フラッシュ型A/D変換

        - あらかじめ2^n-1個に分圧された基準電圧とアナログ信号を比較器（コンパレータ）で同時に比較を行い、比較結果をエンコーダでデジタル信号に変換する
        - デメリット
            - コンパレータが2^n-1個必要
            - 回路規模や消費電力が大きくなる
            - 分解能は8ビット程度が限界
        - メリット
            - アナログ信号を標本化する回路（サンプリング・ホールド回路）が不要
    - パイプライン型ADC
        - 一般的な1.5bit/ステージ構成の場合、MSBを決めるステージ1から順番にパイプライン動作でいくつかの処理を繰り返す
        - デメリット
            - 制御などリアルタイム性が必要なものには向かない
        - メリット
            - 高分解能(16ビット)が実現可能
            - 高速に変換が可能
    - ΔΣ（デルタ・シグマ）型ADC
        - アナログ信号をオーバーサンプリングし、それをΔΣ変調を用いてアナログ信号の振幅に応じた低ビットデータ（例えば１ビット）に変換後、デジタルフィルタで帯域外のノイズ除去とデータの間引きを行うことで本来のサンプリング周波数でのデジタル信号への変換ができる
        - デメリット
            - 応答性が悪い
        - メリット
            - A/Dコンバータ基本形の中で最も高分解能(32ビット)
            - 一般的に逐次比較型と比べ変換速度は遅い

参考

- ADC 基本形1（フラッシュ型）
- ADC 基本形2(パイプライン型)
- ADC 基本形4(ΔΣ型)

感想

特にアンセンブリ言語のデバックを行うときは、どの部分まで動作しているかをしっかりと把握した上で順番に解決していくことが必要だと感じた。今後はデバックの順序をもう少し意識してプログラムや回路の見直しなどのデバック作業を行うようにしたい。