AD/DA 変換

学籍番号 045713C:大城和也

実験実施日　平成 17 年 11 月 07 日 (月)
提出日　　平成 17 年 11 月 14 日 (月)

実験共同者
045709E：上原直久
045734B：知念栄作
045739G：友寄雄一朗

1 実験目的

A/D 変換の仕組みを学ぶとともに，市販の A/D ボードの使用法を習得することを目的とする．

2 実験結果

2.1 実験 1

サンプルプログラムが手元のノートパソコン上で動作することを確認せよ．

2.1.1 サンプルプログラム (修正後)

```
01:  #include <stdio.h>
02:  #include <conio.h>
03:  #include <io32.h>
04:
05:   union _tag{
06:   long l;
07:   char c[4];
08:  };
09:
10:  #define ADR 0x240               /* I/O アドレスの指定 */
11:  #define CHLS 8                  /* チャンネルの数 */
12:
13:
14:   void main(void)
15:  {
16:   int     ModeData = 5;           /* モードの設定データ */
17:   union _tag ClockData = { 799 }; /* クロックの設定データ */
18:   union _tag CountData = { 9 } ;  /* サンプリング回数の設定データ */
19:
20:   int     AiData, i, Count = 0;
21:   float   AiVolt;
22:   outp( ADR+6, 0 );               /* 初期化 */
23:   outp( ADR+6, 1 );               /* サンプリングモードの設定 */
24:   outp( ADR+7, ModeData );
25:   outp( ADR+6, 2 );               /* サンプリングクロックの設定 */
26:   outp( ADR+7, ClockData.c[0] );
27:   outp( ADR+7, ClockData.c[1] );
28:   outp( ADR+7, ClockData.c[2] );
29:   outp( ADR+6, 3 );               /* サンプリング回数の設定 */
30:   outp( ADR+7, CountData.c[0] );
31:   outp( ADR+7, CountData.c[1] );
32:
33:   printf("          ");
34:   for ( i = 0; i < CHLS; i++ )
35:    printf("         ch%d ",i);    /* ch0〜ch7 まで標準出力 */
36:   printf("\n");
37:
38:   outp( ADR+2, CHLS-1 );          /* サンプリングの開始 */
39:   do{
40:          if (inp(ADR+2) &2){     /* バッファ内のデータの有無 */
41:          printf("%5d: ",++Count);
42:                  for( i=0; i < CHLS; i++){  /* 8 回繰り返す */
43:                  AiData = inpw(ADR);  /* データの入力 */
44:                  AiVolt = (float)AiData * 20 / 4096 -10; /* ボルトへの変換 */
45:                  printf("%+7.3fv ",AiVolt); /* 値の出力 */
46:                  }
47:                 printf("\n");
48:          }
49:   }while( inp( ADR+2 ) & 3 );    /* データがある限り繰り返す */
50:  }
```

2.1.2 説明

実験指導書にあるサンプルプログラムはそのままではコンパイルできない．コンパイルするための改善が必要となってくる．

サンプルプログラムからの改善点は，1 つめに 5 行目の"union_tag"となっていた所をスペースを空けて"union _tag"にした．union は共用体の型を示しており，"_tag"がその変数となっている．これをスペースを空けずにすると共用体として見られなくなり，エラーが出力される．2 つめに 10 行目の"#define ADR 0x280"となっていたところを"#define ADR 0x240"にした．これは I/O のアドレスを指すものでこのアドレスの指定を間違えるとコンパイルに問題はないものの A/D ボードが認識さていない状態のプログラムを作ることとなる次にサンプルでは main 関数を終了する'}' が欠けていたので追加した．これらを行うことによりコンパイルできるようになった．

2.2 実験 2

何かキーを押すまで，前チャンネルで 0.1 秒おきに A/D 変換を行い，1 秒おきに表示するプログラムを作成せよ．

2.2.1 プログラム

```
01:  #include <stdio.h>
02:  #include <conio.h>
03:  #include <io32.h>
04:
05:   union _tag{
06:   long l;
07:   char c[4];
08:  };
09:
10:  #define ADR 0x240                /* I/O アドレスの指定 */
11:  #define CHLS 8                   /* チャンネルの数 */
12:
13:
14:   void main(void)
15:  {
16:   int     ModeData = 5;            /* モードの設定データ */
17:   union _tag ClockData = { 999999 };/* クロックの設定データ */
18:   union _tag CountData = { 9 };    /* サンプリング回数の設定データ */
19:
20:   int     AiData, i, Count = 0;
21:   float   AiVolt;
22:   outp( ADR+6, 0 );                /* 初期化 */
23:   outp( ADR+6, 1 );                /* サンプリングモードの設定 */
24:   outp( ADR+7, ModeData );
25:   outp( ADR+6, 2 );                /* サンプリングクロックの設定 */
26:   outp( ADR+7, ClockData.c[0] );
27:   outp( ADR+7, ClockData.c[1] );
28:   outp( ADR+7, ClockData.c[2] );
29:   outp( ADR+6, 3 );                /* サンプリング回数の設定 */
30:   outp( ADR+7, CountData.c[0] );
```

```
31:   outp( ADR+7, CountData.c[1] );
32:
33:   printf("       ");
34:   for ( i = 0; i < CHLS; i++ )       /* ch0～ch7 までを標準出力 */
35:    printf("     ch%d ",i);
36:   printf("\n");
37:
38:   outp( ADR+2, CHLS-1 );             /* サンプリングの開始 */
39:    do{
40:          if (inp(ADR+2) &2){        /* バッファのデータの有無 */
41:                  Count++;
42:                  if(Count % 10 == 0 ) printf("%5d: ",Count/10);
43:                  for( i=0; i < CHLS; i++){
44:                     AiData = inpw(ADR);    /* データの入力 */
45:                     AiVolt = (float)AiData * 20 / 4096 -10;
46:                     if( Count % 10 == 0 ){ /* Count が 10 の倍数 */
47:                          printf("%+7.3fv ",AiVolt); /* 値を出力 */
48:                     }
49:                  }
50:                  if(Count % 10 == 0 ){ printf("\n");
51:  }               }
52:    }while( (inp(ADR+2 ) & 3) /*&& (!kbhit())*/ );
53:   outp(ADR+6,4);                      /* サンプリングの停止 */
54:  }
```

2.2.2 実験結果

以下の図 1 は上記のプログラムを動かした際の実行結果である.

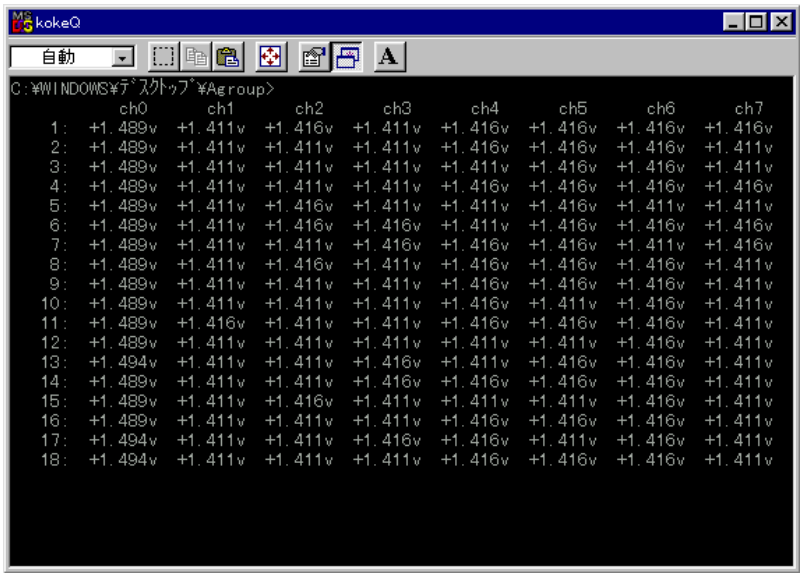

図 1: コマンドプロンプト

2.2.3 説明

このプログラムはサンプリングクロックの設定を変えることでA/D変換の割合を0.1秒に一度にした．サンプリングクロックの設定は17行目でClockDataにより行われている．ここで扱われているClock Dataは以下の式の基に導かれる．但し，SamplingClockはnsec単位．

$$ClockData = \frac{SamplingClock}{100} - 1$$

上記の式より，今回は0.1秒を設定すればよく0.1[s]は10^8[nsec]であるので

$$\begin{aligned} ClockData &= \frac{100000000}{100} - 1 \\ &= 999999 \end{aligned}$$

という値が導きだされる．

次に1秒ごとに表示する方法はループによって実現した．サンプルのプログラムでは変換する度に値を出力されていた．今回は0.1秒で変換を行っていたので1秒毎に画面に出力するには10回変換されたときに値を出力するようにすればよい．これは42行46行50行のif文で行っており，Countという変換が行われた回数を数えている変数が10で割れるなら画面出力という形になっている．

最後にdo〜while文による変換と出力の繰り返しから抜けるため，つまりプログラムを終了するために，条件として!kbhit()というものを加えた．kbhit関数はキーボードが押されているかどうかを調べる関数で入力がない場合は"０"が返され，入力があった場合には"0"以外を返す．よって!kbhit()というふうにすることでキーボード入力がない限りループするプログラムとなっている．ちなみにもう１つの条件である`inp(ADR+2 ) & 3`は変換の対象となるデータが無くなった時などに停止するようになっていると思われる．

実行結果を見ると出力されているボルトはほとんど一定である．これは今回の実験でアナログのデータの入力を行わなかったためだと考えられる．

3 A/D変換方式について

アナログの情報をデジタルに変換するものはA/D変換器と呼ばれるが，A/D変換をする方法には今回の実験で扱った逐次比較型以外にも多数の変換方式がある．それぞれの変換方式は採取するデータの性質やコストなどによって選択されている．以下に逐次比較型以外の代表的な変換方式としてフラッシュ型，積分型，二重積分型．デルタシグマ型について説明する．

3.1　フラッシュ型

フラッシュ型は並列比較方式とも呼ばれ，逐次比較方式と同じく電圧比較型の変換方式である．A/D 変換にはコンパレータという比較を行う回路が含まれており，コンパレータは 2 つのアナログ入力と，1 つのデジタル出力がある．コンパレータはアナログ入力の一方を基準とし，他方がその基準より大きいか低いかを調べ結果をデジタルの 0 か 1 で出力する．フラッシュ型はこのコンパレータを単純に並列に繋げ，値を分割して比較することで数ビットの変換を行い最後にデコーダにかけて 2 進数にすると行ったものである．

利点は並列処理による高速な変換が可能なことで，欠点はコンパレータを多用するため回路面積とコストがかかるとこである．

3.2　積分型

積分型は時間比較型の変換方式で，積分器に一定の直流電圧を入力したときその電圧は積分時間に比例する．コンパレータには一定の電圧とこの積分回路からの出力が入力されており，積分器の出力が基準となる電圧を超えたときにコンパレータの出力が反転する．このときにパルス計数も停止し，そのパルスの数が計測される．これがディジタルとしての値となり，A/D 変換される．この一様の動作はコンデンサに一定の時間で充電する動作と同じである．

図 3 はそれらの様子を表している．In は電圧，xp1 が積分器，xp2 がコンパレータ, Counter がパルス波をそれぞれ表している．積分型の利点はパルス

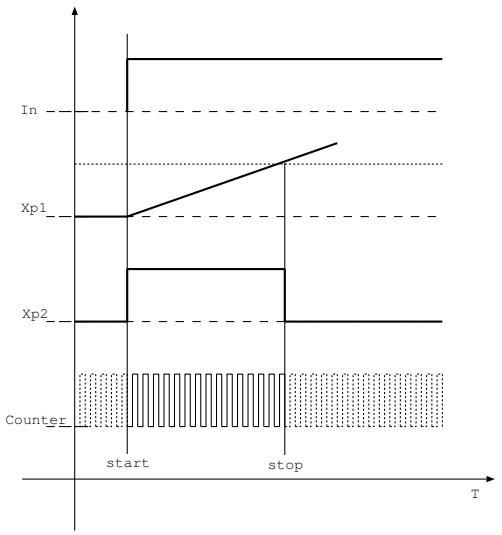

図 2: 積分型

の密度を増やすことで変換の精度を高めることが可能なことだが，積分がただとノイズなどの誤差要因があった場合，その時の分まで積分回路に入るので誤差がでやすい．これが欠点である．

3.3 二重積分型

二重積分型は先ほど説明した積分型を改良して，誤差の欠点をなくしたものである．変換の方法は初めに積分型と同様に電流を一定時間積分する．二重積分ではこの時のパルスを数えるのではなく，次に入力と逆の極性で積分するときにパルスを数える．このパルスの数がディジタルな変換値となる．この動作は一定の時間でコンデンサを充電し，その放電時間を計っているのと同じである．

図 3 はこれらの様子を表している．

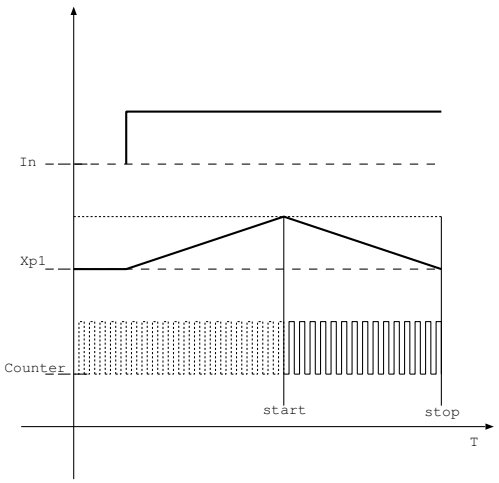

図 3: 二重積分型

二重積分型の利点は積分型同様に精度が高いということ．一方，欠点は変換速度が遅いということである．

3.4 デルタシグマ ($\Delta\Sigma$) 型

デルタシグマ型は時間比較型の一種で，微分と積分を扱って変換を行う．デルタシグマはデルタコンバータというものが元となっており，その動作は前の値より増加したら 1，減少したら 0 というように 1 ビットで値を表現するものである．これは値が変化しない時の測定ができない．デルタシグマでは積分回路を組み込むことで値が変化しない時でも対処できるようにしている．

デルタシグマはまた，精度を上げる (量子化誤差を減らす) ために本来のサンプリングデータの間に，いくつもの計算した値を挿入するオーバサンプリングを行っており，これによってサンプリングの速度を速くしたのと同様な効果を得ている．

デルタシグマ型の利点は精度が高いことと，処理が 1 ビットで済み，処理のほとんどをデジタルで行うことにより，IC 化することが容易でさらにコストが安いということである．一方，欠点は変換が遅いということ．特にオーバサンプリングを行っているので遅延が発生する．

4 感想

今回の実験は思ったより早くレポートが仕上がりました．といっても前日になりますが … 。A/D 変換には色々な種類があることが調査をすることで分かったのですが，なかなかに難しくてその方法について詳しく分かったどうかは自分でも怪しげです．実験ではプログラムの変更に少し戸惑いました．特に I/O のアドレスの辺りは全然気付きませんでした．後，実験した後に気付いたんですが自分たちの班，画像を間違えて実験 (2) のやつを二枚取ってたので実験 (1) の結果がなかったです … 。これは今回の一番の反省点でした．次からは気をつけようと思います．

参考文献

[1] AD 変換機
”http://www.comb.kokushikan.ac.jp/lecture/envmeasure/node58.html”

[2] 測定器玉手箱
”http://www.orixrentec.co.jp/tmsite/index.html”

[3] 失敗しない温度計測豆知識
”http://www.ondokeisoku.jp/shinan/mame2.html”,

[4] AD12-8(PM) 解説書, 出版:CONTEC