パソコンによる計測・制御システムの構築

電気・情報系（情報工学科）　岡　和喜男

1　はじめに

計測（Instrumentation）と測定（Measurement）は、はかる意味で使われる同義語である。計測とは、測定値をある尺度から作られる基準値を比較して客観的数値を用いて表す操作である。パソコン計測・制御での計測とは、はかることに関連する事柄、はかる方法、はかる過程など、測定よりも広い意味で使用されている。また、制御とは、ある目的に適合するように、対象物に操作量を加えることであると定義されている。著しいパソコンの高性能化が進んでいる現在、パソコンによる計測・制御システムも進化、高機能化している。複数の計測機能を混在できる柔軟性の高さ、収集後のデータ加工のしやすさ、市販の表計算ソフトや解析ツールを使ったデータ分析を、計測している同じプラットフォームでの処理、操作性のよさ、ユーザーによるカスタマイズのしやすさなどが、これからのパソコンによる計測・制御システムに求められている。

現在、学生実験の一部である計測実験の「計測・制御システム」をMS－DOSからLinuxに移行することになり、Linux用のシステムとして再構築中である。今回、この再構築中の「計測・制御システム」について報告する。

2　Linuxの導入

本システムは、学生実験の一部であり、実際に機器を操作し、得られた技術やデータを理論学習と比較することで、より一層の学習効果を求めている。ここでは実際に、計測・制御するためのプログラミングを行い操作している。そのためには、プログラミング用の言語ソフトとコンパイラー（機械語翻訳ソフト）とOSソフトが必要となる。市販のソフトを購入すると１台につき数万円になり、学生実験のように数十台の場合は、簡単に数百万円の経費となる。また、更新のたびに経費がかさむことにもなる。Linuxの場合は、OSはフリーソフトであり、UNIXのソースライセンスをいっさい使わずに、UNIXとほとんど同じことができるOSである。このOS上で実行できる、C言語ソフトとgccコンパイラーもフリーソフトであり、再構築のシステムとしてLinuxを導入することにした。

3　計測用モジュール

モジュールとは、組み合わせて一つのハードウェア・システムを作ることができる基本単位で、それ自体高い機能を有する。プログラムの中では、論理的にそれだけで完結し

た部分を指し、プログラム自体をモジュールの集まりとみなすことができる。計測モジュールには、ISA（Industry Standard Architecture）や PCI（Peripheral Component Interconnect）といったパソコンの汎用バスを用いるものと，GP-IB や RS-232C などで外部の専用モジュールに接続するものがある。バスとは、計算機からデータを転送するときに、一度に 16 本／32 本（64 本）といった多数の線を使って、大量のデータを転送する、回線や端子をバスと呼び、現在マザーボード上の外部接続用に用いられている。PCI は 32bit、ISA は 16bit のバスである。再構築にさいしては、PCI バスの拡張ボードである、ＤＡ変換ボード（ディジタル量をアナログ量に変換する装置）と GP-IB インターフェイスボード（アナログ量をディジタル量に変換する外部専用モジュール装置をディジタルで制御できる装置）を使用している。

３．１　ＤＡ変換ボード

パソコンで計測・制御するには、外界と信号をやり取りしなければならない。コンピュータが用いる量はディジタル量であるのに対し、速度や温度などはアナログ量である。この両者を操作するとき、多くの場合、ＤＡ変換／ＡＤ変換を経ることになる。本システムでは、I/O ポートアドレスマップが公開されている、インターフェス社の 12 ビット分解能のＤＡ変換ボード PCI-3329 を用いＤＡ変換の試験操作を行った。

３．２　ＤＡ変換ボードの使用法

図１に公開されている PCI-3329 の I/O ポートマップの構成を示す。ADR＋0 から ADR＋F までの１６ポート占有する専用モジュールである。

ＡＤＲ＋０、＋１ｈ：DA 変換データ
ＡＤＲ＋２ｈ：ＤＡチャンネル設定
ＡＤＲ＋３ｈ：インターバルタイマ（弊社標準）
ＡＤＲ＋４ｈ：極性反転
ＡＤＲ＋５ｈ：割り込み設定
ＡＤＲ＋６ｈ：タイマゲート制御
ＡＤＲ＋７ｈ：ボード識別用ＲＳＷ
ＡＤＲ＋８、＋Ａ、＋Ｃ，＋Ｅｈ：プログラマブルタイマ（８２５４相当）
ＡＤＲ＋Ｆｈ：シリアルＥＥＰＲＯＭ制御
ＡＤＲ＋９ｈ：デジタル入出力制御
ＡＤＲ＋Ｂｈ：アナログ出力切替え制御

図１　I/O ポートマップの構成

試験操作のため最低限のポートアクセスで動作するようにプログラミングを行い操作する。例えば、図２に示す ADR＋B のポートアドレスは、１６進数記述されているので、１０進で表すと基本 ADR から 11 番目のポートの書き込みレジスタ（出力レジスタ）である図中

のbitに1を記述する、すなわち「00000001」なるディジタル信号を送り込むことによって、アナログ出力電圧の出力を可能にする。言い換えると、このモジュールの仕様に見合った信号を送ることで一つの仕事が完了していることになる。ただし、パソコンの電源が切れた場合などは、フラグ（bitに1を立てる状態）を維持できないため、再度、信号を送りフラグを立てる必要がある。

ＡＤＲ＋Ｂｈ：アナログ出力切替え制御

I/Oポート アドレス	入出力	内　　容
ADR+Bh	入力	未使用
	出力	アナログ出力切替え制御 bit 7 6 5 4 3 2 1 0 － － － － － － － OSEL

電源投入時：００ｈ

■ＯＳＥＬ　　アナログ出力切替え制御

０…アナログ出力は０Ｖ固定（全ＣＨ）

１…アナログ出力は各ＣＨのＤＡ出力

電源投入時にはＯＳＥＬ＝０に初期化されるので、アナログ出力電圧は全ＣＨ０Ｖになります。その後ＤＡ出力させるにはまずＯＳＥＬ＝１にする必要があります。１回 ＯＳＥＬ＝１にしたら電源を切らない限り再設定する必要なし。

図２　ADR+Bの I/O ポートマップ

同様に、実際に電圧を出力する場合は、さらに、選択 ch ポート（ADR＋2）と図３に示す12bitに分解したDA変換データの出力ポート（ADR＋0、1）に、モジュールの仕様に見合った信号を送ることが必要になる。

DA変換データ（ADR+0h、+1h：出力）

ADR+1h								ADR+0h							
－	－	－	－	B11	B10	B9	B8	B7	B6	B5	B4	B3	B2	B1	B0

【バイポーラ時】

スケール	出力電圧レンジ ±X(V)	DA変換データ B11 ～ B0	HEX
＋フルスケール	＋X－1LSB	111111111111	FFFH
：	：	：	：
0	0	100000000000	800H
：	：	：	：
－フルスケール	－X	000000000000	000H

【ユニポーラ時】

スケール	出力電圧レンジ 0～＋X(V)	DA変換データ B11 ～ B0	HEX
＋フルスケール	＋X－1LSB	111111111111	FFFH
：	：	：	：
ハーフスケール	＋X／2	100000000000	800H
：	：	：	：
0	0	000000000000	000H

【対応表】

入力電圧レンジ(X)	1LSB当たりの電圧(mV)
0～＋5	5／4096≒1.2207
－5～＋5	10／4096≒2.4414

図３　ADR＋0、１の I/O ポートマップ

３．３　ＤＡ変換ボードLinux 用のI/O ポートプログラミング

C プログラムから I/O ポートを使う方法の一例を示す。I/O ポートをアクセスするためのルーチンとしてLinux の場合は、/usr/include/asm/io.h の中でインラインマクロとして定義されている。記述は、#include <asm/io.h>である。ただし、これらのルーチンを使うソースでは、少なくとも gcc 2.7.2 またはそれよりも前のバージョンを使う場合に制限がある。通常、危険防止のためポートのアクセス許可は、プログラムに与えられていない。例えば 0x000 から 0x3ff までのポートに対するアクセス許可は、ioperm()関数、全てのポート(65536 個)に対するアクセス許可は、iopl()という関数を呼び出し許可を与えなければならない。この機能は時としてシステムの破壊をもたらす危険度高い関数である。また、インラインマクロとして定義された中に、ポート(port)への入出力のための低レベル関数として、outb () inb () を呼び出すことができる。プログラミングの際には、使用するPCI-3329 のポートが65536 個の何処のポートにどの様にマッピングされたかは、OS をインストールする際に決定されるため、調査してから関数を利用する必要がある。Linux コマンドの中に、システムの全ての PCI バスとそこにつながっている全てのデバイスに関する情報を表示するユーティリティがある。これらのユーティリティを使いPCI-3329 の個所を抜粋した一例を下記に示す。

```
>cat /proc/pci
Bus  1, device   9, function  0:
Hot Swap Controller: Unknown vendor Unknown device (rev 1).
Vendor id=1147. Device id=d01. Medium devsel.  IRQ 10.    I/O at 0x1420 [0x1421].
> lspci -vvx
Class ff00: Interface Corp: Unknown device 0d01 (rev 01)
Subsystem : Interface Corp: Unknown device 0001
Control   : I/O+ Mem- BusMaster- SpecCycle- MemWINV- VGASnoop- ParErr-
Stepping- SERR- FastB2B-
Status    : Cap- 66Mhz- UDF- FastB2B- ParErr- DEVSEL=medium >TAbort- <TAbort-
<MAbort- >SERR- <PERR-
          Interrupt: pin A routed to IRQ 10
          Region 0: I/O ports at 1420
```

下記は、システムに書き込まれダンプ内容を示す。

```
  00: 47 11 01 0d 01 00 00 02 01 00 00 ff 00 00 00 00
  10: 21 14 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  20: 00 00 00 00 00 00 00 00 00 00 00 00 47 11 01 00
  30: 00 00 00 00 00 00 00 00 00 00 00 00 0a 01 00 00
```

得られた調査結果を基に、PCI-3329 のＤＡ変換ボードから、出力電圧５Ｖと 10V を 2 秒と 1 秒刻みに出力させると同時にグラフ化しデータとして保存する試験用プログラムを作成しＤＡ変換の試験操作を行った。作成したプログラムの一例は付録に示す。なお、時間割り込みのモジュールについては、本体パソコンのシステムクロック周波数（cpu ： 598.067 MHz）に狂いが生じるか検査を兼ね、直接ポートから操作する方法と内部関数で操作する方法を用い、比較するようプログラミングしている。

３．４　試験結果

図４は、ＤＡ変換ボードが組み込まれたパソコン画面からプログラムを動作する様子である。この例では、3 個のプロセスを動作させ、他のプロセスに影響が有るか調査している。試験操作のプログラムを動作させ擬似的ではあるがリアルタイムに表示されたグラフ画面の結果を図５に示す。また、同時にＤＡ変換ボードから実際に出力されている電圧を岩通製デジタルストレージオシロスコープ 2V/2sec(DIV)で測定した結果を図６に示す。図６から、意図した時間変化に対して電圧変化していることが認められる。また、時間割り込みによるプロセスの影響はなく意図した電圧を制御できることが認められた。

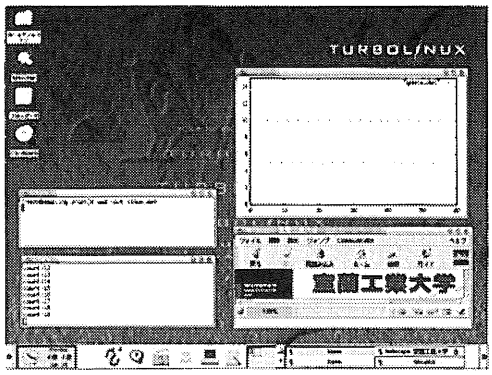

図４　パソコン画面

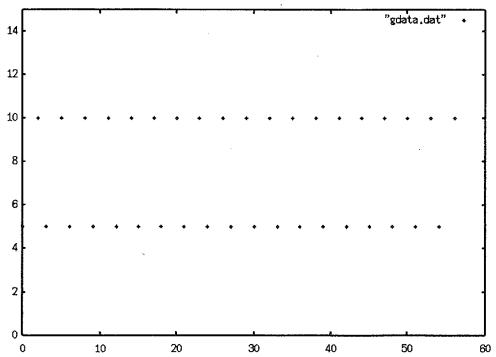

図５　グラフ画面

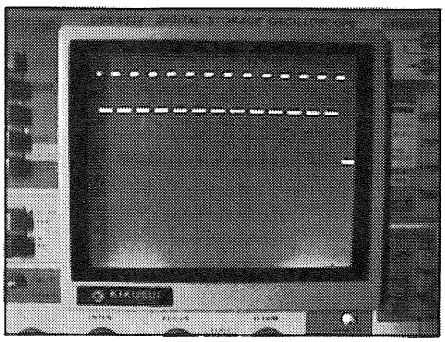
図６　測定結果

４　おわりに

本報告の内容は、パソコンによる計測・制御のＤＡ変換ボードの使い方に重点が傾いてしまった。計測で使われるデータ取り込み部分である GP-IB インターフェイスボードに関

しては、現在作成中であり、本報告に記載することができなかった。また、Linux については、情報工学科　技官室の方々に設定して頂いた。筆者は、Linux システムを知らないまま利用し、何度かクラシュさせている。その都度、技官室の方々助けられ、プログラミングに没頭できたことを改めて深く感謝する。

付録　試験操作用プログラム例

```
/***  ＤＡ変換用試験用プログラム 2001.1.22 by csse-o  ***/
#include <stdio.h>
#include <asm/io.h>
#include <time.h>
#define GNUPLOT_PATH "/usr/bin/gnuplot"
#define BASEPORT   0x1420 /*  5152(10 進数) */
void clock_wait1(void)
{
        usleep(1000000); /* 1000000us = 1000ms スリープさせる */
}
void clock_wait2(void)
{
        double t;
        for(t=0;t<3000000;t++)/* 3000000 回 = 2000ms  ディレイさせる */
        { outb(1,0x80);}
}
/* 時間設定、グラフ設定を含めたメイン関数   */
int main()
{
        FILE *plot,*plot_data;
        time_t start_t,over_t,over_t2;
        int i;
        double t;
        char *dat1 = "plot [0:120] [0:20] ¥"gdata.dat¥"¥n";
        char *dat2 = "replot ¥n";
        plot_data = fopen("gdata.dat", "w+");
        plot = popen(GNUPLOT_PATH, "w");
        if(plot == NULL){
        fprintf(stderr, "Oops, I can't find %s.", GNUPLOT_PATH);
        exit(EXIT_FAILURE);
        }
```

```
        fprintf(plot, dat1);
        fflush(plot);
    getchar();
        iopl(3);
        outb(0x1,0x142b);/*   5163   outb(0,BASEPORT) */
        outb(0x0,0x1422);/*   5154                    */
        outb(0xf,0x1420);/*   5152                    */
        outb(0xf,0x1421);/*   5153                    */
        time(&start_t);
        for(i=0;i<15;i++)
        {
                printf("count:%d¥n",i);
                outb(0xf,0x1420);/*   5152         */
                outb(0xc,0x1421);/*   5153         */
                        //printf("status :  %x¥n",inb(0x1420));
        time(&over_t);
                fprintf(plot_data,"%ld %d ¥n",over_t - start_t,5);
                clock_wait2();
                outb(0xf,0x1420);/*   5152          */
                outb(0xf,0x1421);/*   5153          */
        time(&over_t2);
                fprintf(plot_data,"%ld %d ¥n",over_t2 - start_t,10);
                clock_wait1();
                fprintf(plot, dat2); /* reploter */
                fflush(plot);
                fflush(plot_data);
        }
        /* 出力をＯＶにして終了する */
       outb(0x0,0x1420);/*   5152          */
        outb(0x8,0x1421);/*   5153          */
        getchar();
        fflush(plot_data);
       fclose(plot_data);
        exit(0);
}
/* end of daout.c */
```