MS-Windows XP 上でのプロセス制御用リアルタイムシステム

立花 康夫[1]・河合 敏勝[1]・光田 明史[2]
1 電気電子工学科
2 平成14年度電気電子工学科博士前期過程

A Real Time System for the Process Control on MS-Windows XP

Yasuo Tachibana[1], Toshikatsu Kawai[1] and Akichika Mitsuta[2]

Abstract

In this paper, we propose an example of the construction of a digital control system on the DOS-V PC and the general purpose OS. And we expect useful effect for the education of the control system by making and operation of the proposed controller. Inserting the timer and analog to digital transform boards into PCI slots in the DOS-V PC, we can construct a real time environment. In a computer, we realize the proposed controller as an object oriented program using C++ language. The control objective is a keeping heat system constructed by an incandescent lamp. The actuator of our presented system is a triac current control system driven by MDP(Motor Driven Potentiometer). In this circumstance, we are able to acknowledge the auto-manual transfer, bump-less transfer, anti-reset windup, fail safe environment, and the easiness of the construction of control algorithms. The presented system is operated on the Pentium IV(1.6GHz) CPU and MS-Windows XP OS. Also this system is running stably on the 10msec sampling period and 100msec control period. Now we are using this system for the purpose of the understanding of a controller and the basic system for the study on the graduate course. And we can show that our proposed system has useful effects for the education of control system.

Key Words: DOS-V, MS-Windows XP, Real Time System, Object Oriented Programming, C++, Digital Control, MDP

1. はじめに

この論文は、大学での制御工学の授業あるいは、卒業研究での課題検討段階での制御系の題材として、汎用のパソコンによる制御系構成例を与え、その作成と運用により有効な教育効果が得られることを示す。

制御工学は、工学全体の設計思想を与えるものであるため、勢い、抽象化された概念、特に数学的な説明が多くなり、どうしても授業内容は難しいという印象を与える。そのため、具体的な対象の制御の例や、具体的な制御系を示すことにより、理解を深めて行くようにすることが強く要求される。このうち、具体的な対象の制御の例は家電品、自動車、工業プラントあるいはロボット等と数多く挙げて説明ができる。しかし、具体的な制御系の例を示すのは意外に難しい。それは、最近の制御系の具体的な実現手段が殆どμプロセッサ等の計算機であり、システムの中にさりげなく組み込まれているからである。そして、制御系の実態を説明するにはどうしてもμプロセッサ等の計算機システムの説明もしなければならないが、制御工学という範囲ではそこまで手を広げて説明することは難しい。

計算機について言えば、我々は、研究室に、いわゆるDOS-V(IBM-PC)パソコンを卒業研究や1年生と3年生ゼミ等の目的のためにたくさん所有している。これらのオペレーティングシステムはMicrosoft社の（以後、MS-で示す）MS-Windows XP(2000等もある)で、全てLANで接続されている。また、これらのパソコンの利用に関しては、関連する授業もいくつか用意されている。特に、MS-Visual BasicやMS-Visual C++等の言語によるプログラム作成について学生はある程度まで経験している。そして、殆どの学生は、家でもこれらを所有していて、使用に慣れている。

このような背景を考慮すると、汎用のパソコン上で、ごく一般的なプログラムツールを使用して、プロセス制御系を構成することができれば、計算機の一般的な説明は不要となり、プログラムの作成、他のソフトウエアとの連携、そしてLAN等の通信システムの利用等の点を考えて大変に有効であることがわかる。

この論文では、これらの汎用パソコンのPCI(Peripheral Component Interconnect local bus)スロットに、2組のプロセスIO(Input Output)ボードを差し込んで、リアルタイム環境を構成し[1],[2],[3]、そこでの制御系実現の例としてPID(Proportional Integral Differential)制御系の実施例について示す。制御系の具体的なプログラムは、Borland社のC++ Builderによりオブジェクト指向プログラムとして実現している[4]。また、制御対象は、教育的に分かり易い対象であることが望まれることを考え、白熱電球を熱源に用いた保温系とした。また、アクチュエータとしては、MDP(Motor Driven Potentiometer)で駆動されるトライアック電流制御回路を用いる。

このような構成により、検討過程にある各種制御アルゴリズムを容易に実現することができ、制御実行と同時に他の汎用ソフトを平行して使用でき、さらにLANによる通信機能を利用して複数のパソコンとの制御的なリンクが可能となる。制御系の一般的な機能としては、自動運転（自動制御状態）と手動運転との切り替え、特に手動から自動に切り替えた際のバンプレストランスファー(Bump-less Transfer)の実現、積分制御要素の飽和の解消を行うアンチリセットワインドアップ(Anti Reset Windup)等は、簡単なプログラムで実現でき、学生の理解の上で極めて有効である。また、アクチュエータはMDPを含む外部アナログ回路としたためフェールセーフの機能が実現できる。このように、制御系の有すべき基本機能をわかりやすく、C/C++等の汎用言語で提示することができ、その動作を現実の制御を実施しながら体得することができる。

以下、システムの構成と具体的な検討点について述べる。

2. リアルタイム処理

制御系や信号処理系を構成する上で最も重要な事項は、正確なリアルタイム性を確保することである[5]。すなわち、これらのシステムは自然現象を相手にしているから、それらの間の同期は正確な時間で実施されなければならない。具体的には、対象の信号をサンプル周期T_sでサンプリング

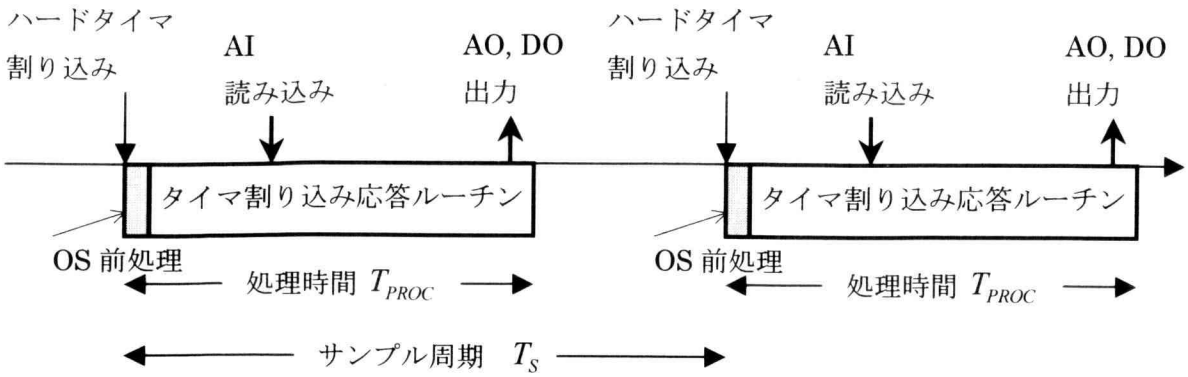

図 1　処理の時間的な流れ

し、ディジタル処理をして、同じ周期内で対象に出力を戻さなければならない。このような一定周期での処理を実施するにはOSの時間に関するサービス（Time Scheduling）を用いることになる。

具体的には、Visual Components(Borland C++ Builder の Visual 環境を構成する部品、MS-Visual Basic では、Visual Controls という。以後、C++ Builder の用語を使用する)の中の Timer を用いると OS よりこのサービスを受けることができる。Timer には、基本となる 2 つのプロパティ Enabled と Interval がある。Enabled を true にすれば、しばらくして(OS の処理時間後)Timer のイベントハンドラ（割り込み処理ルーチン）へ処理が移る[4]。さらに Interval に指定した時間の後、このイベントハンドラの動作が再開される。

このように OS の用意した時間サービスを簡単に受けることができるが、残念なことに、PentiumIV(1.6GHz)程度でもこの周期は数 10msec が限度であり、しかも、OS の負荷の状況によっては変動が激しい(いわゆる、タイムジッタが発生する)。したがって、この Timer によっては、制御系や信号処理系を実現するのは難しいと言える。我々は、便宜上、今述べた標準で用意されているこの Timer を**ソフトタイマ**と呼ぶことにする。

そこで、我々は、正確な時間間隔で、外部割込みを発生させ、その割り込み応答ルーチンで、正確な時間の同期をとる方法を用いることにする。PCI スロットに差し込む各種のプロセス IO ボードが例えば Interface 社より発売されている。我々は、この中のタイマボード PCI-6103[1]を用いる。また、アナログ入力とディジタル出力 2 ビットを併設する AI ボード PCI-3170A[2]も用いる。以下に説明するように、対象系への出力を与えるアクチュエータは MDP で駆動される電流制御器であるが、この MDP は 2 ビットのディジタル出力で駆動する。このタイマボードは先のソフトタイマと区別するために**ハードタイマ**と呼ぶ。ハードタイマは T_s = 1 μ sec～3.6028×10^{10}sec の範囲で設定できるが、実際に MS-Windows XP 下で利用できるのは、$T_s \geq 1\,m$sec であろう。ハードタイマより発生する外部割込みにより**割り込み応答ルーチン**（我々

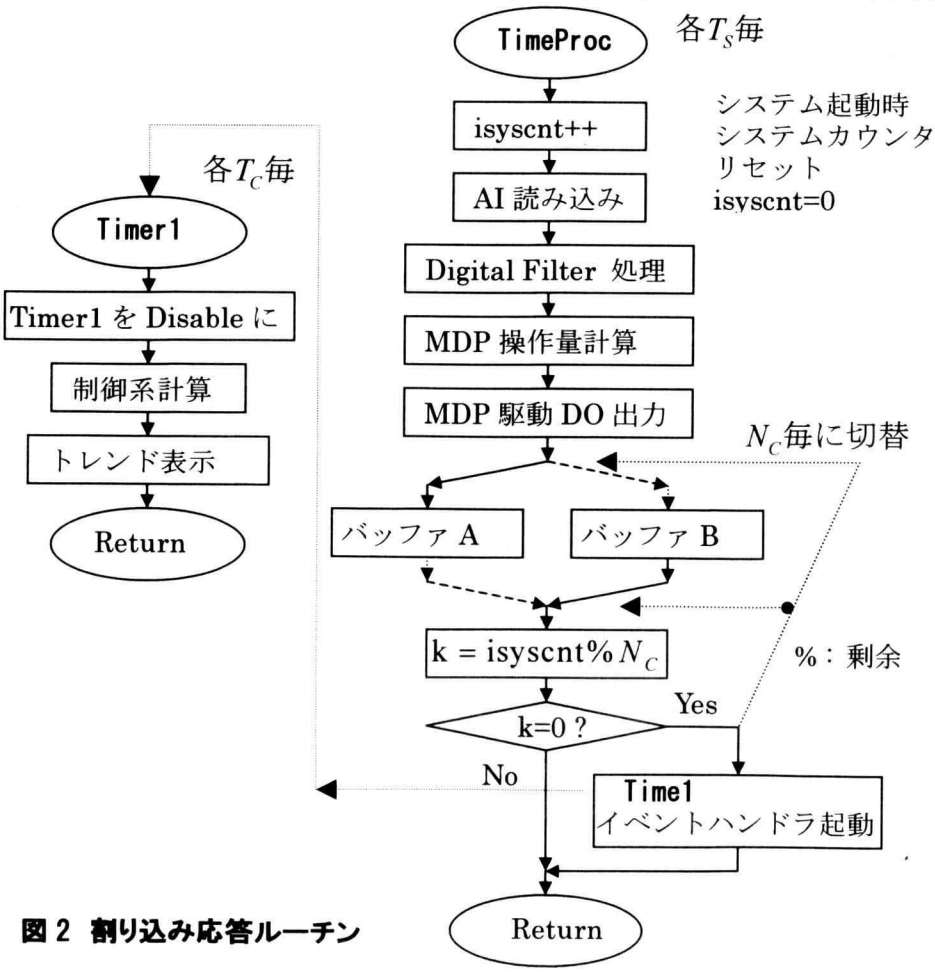

図 2　割り込み応答ルーチン

はこのプログラムを **TimeProc** と名づける）を起動し、そこでアナログ入力とディジタル出力を実施する。従って、我々が構成した制御系のためのリアルタイム環境における処理の時間的な流れを図示すると**図 1** のようになる。このようなリアルタイム環境では $T_s = 1\,m\,\mathrm{sec}$ でも十分に正確に稼動させることができる。

ところで、実際の制御系の動作は、制御周期 T_c で実施される。対象が熱系等の場合には、T_c は信号のサンプル周期 T_S に比べてかなり大きな値でもよい。すなわち $N_c = T_c / T_s \gg 1$ とできる。そうならば、$T_s = T_c$ とすればよい様に考えられるが、計測信号のフィルタリングや MDP の制御のためには T_c よりずっと小さい周期が必用なのである。また、系の状態信号（設定値、制御量、操作量等）は時々刻々グラフ表示（トレンド表示）する必要があるが、これは、T_S でサンプリングした値を T_c 毎に表示しても十分である。

そこで、割り込み応答ルーチン **TimeProc** では、N_c 毎に、制御系とトレンド表示系を起動する。具体的には、ソフトタイマ **Timer1** を起動する。すなわち Timer1 の Enabled を true にする（Interval は例えば 0 としておく）。**TimeProc** は一連の処理を実施すると **Enabled** を **false** として自律的に周期的動作をしないようにする。これでソフトタイマ **Timer1** が **TimeProc** に同期して動作する。これらの処理の概要は**図 2** のフローチャートに示されている。図の中で、バッファ A と B は計測データを N_c 組分蓄えることができ、T_c 回毎に切り替わる。**Timer1** ルーチンでは、このバッファの値を利用してトレンド表示を行う。

3. 制御対象

制御系はわかりやすい事を主眼に置き、白熱電球を熱源とする保温器である。**図 3** はその概観を示す写真で、2 枚の耐火ブロックを平行に置きその上部には銅版の蓋をしてある。他の二つの側面は開いている。空間に 10W の白熱電球を熱源としてはさみ、その上部には、K 型熱電対の一端を差し込んである。これにより、この小さな空間の温度を測定している。熱電対の他端は氷水を入れたポットの中に入れ、基準温度として 0℃を使用する。

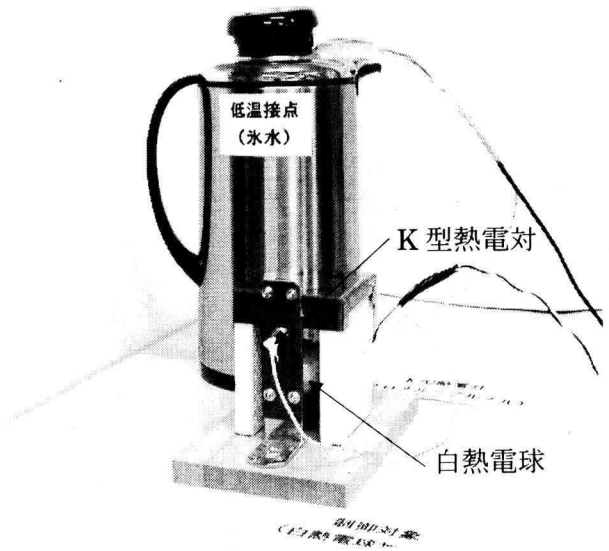

図 3 制御対象（白熱電球保温器）

この系では白熱電球の電流を一定にしても、温度は一定のままにはならない。空間の温度が上昇してくると、空いている側面から冷たい空気が入り込んできて、今度は逆に温度が下がりだす。そしてその空気の温度が上昇を始める。このような変化を繰り返すのである。すなわち、極めて非線形性の強い系であるため線形近似することは困難である。

また、この設備の置いてある部屋は通常の研究室である。夏とか冬では空調が効いている。空調の風が部屋の中を循環し始めると、この装置の空間の温度が変化する。これは、外界からの外乱として捉えられる。

いずれにしても制御系が無ければ、保温器の空間の温度を一定に保つことは困難である。

4. 制御系の動作

制御系は、すなわち、パソコンの中で実施する。システム全体を描くと、**図 4** のようなブロック図で示される。制御対象は **3 節**で述べた白熱電球による保温器である。従って、対象系への実操作量は白熱電球に流れる電流であるが、実際はその設定値である後述 MDP のポテンショメータの電圧 $u(t)$ である。また、空間の温度を $y(t)$ とすれば、対象系への入力は $u(t)$ で出力は $y(t)$ となる。

図 4 でディジタル系は点線で囲んだ部分であり、その動作周期は T_c である。空間の温度 $y(t)$ は K 型熱電対、直流増幅器、雑音除去フィルタ等を介して AD コンバータによりディジタル信号となる。この AD コンバータの動作周期は既に述べたように T_S であり $N_c = T_c / T_s$ は 1 よりかなり大きな数である。そこで、T_S と T_c の差異を利用してディジタルフィルタ等の処理を行い、等価的にはサンプル周期 T_c でのディジタル信号 $y_k = y(kT_c)$ となる。T_c 時間ごとの温度設定値 r_k と y_k の差に基づいてディジタル制御系が操作量 u_k を計算する。すなわち、T_c 周期で発生する u_k に対して、アクチュエータにより周期 T_S でポテンショメータの電圧 $u(t)$ が u_k を 0 次保持した階段関数に一致するように働く。実際にはアクチュエータより周期 T_S で出力されるのは、次節で述べるように 2 ビットの On/Off 信号であり、これにより MDP のモータを回転したり停止したりする。モータに直結したポテンショメータの出力 $u(t)$ で点弧角が定まり、トライアックが動作して電球に流れる電流が定まる。

制御系は、この論文では PI 制御としてある。すなわち、

$$e_k = r_k - y_k \qquad \text{Deviation} \qquad (1)$$

$$w_k = w_{k-1} + e_k \quad w_0 = 0 \qquad \text{Integration} \qquad (2)$$

$$u_k = pe_k + qw_k \qquad \text{P+I Controller} \qquad (3)$$

により計算する。ここで、p, q は制御パラメータである。

5. アクチュエータ

既に述べたように、制御周期 T_c 毎に設定値 u_k が計算される。この値をアナログ信号 $u(t)$ に変換するメカニズムが**アクチュエータ**である。端的に言えば、DA コンバータを周期 T_c で動作させれば片付く話のような気がするが、実際には、そのようにはしていない。それは、もし何かの時に制御系の電源が切れてしまえば、DA 変換器により $u(t)$ を作っ

ていたのでは、突如としてその値が消失する。この際には、対象に加わっている信号 $u(t)$ により、系は大きく乱されてしまうことになる。これでは、フェールセーフの機能を持つとはいえない。

フェールセーフ機能を持つようにするには、信号 $u(t)$ を何らかの機械系に記憶させておく必要がある。機械系に記憶させると、制御系の電源が切れても $u(t)$ は失われず、保持されたままになる。具体的には MDP(Motor Driven Potentiometer)を用いる。図 5 のように MDP は直流モータの軸に直結してポテンショメータを結合した装置である。モータを回転するとポテンショメータの角度が変化する。ポテンショメータの角度により発生する電圧 $u(t)$ をトライアック回路により電流信号に変換する。ポテンショメータの角度は機械的な位置であるから、モータの駆動電源や駆動信号が停止してもその角度は保持されている。これがフェールセーフに用いられる所以である。

トライアックは双方向のサイリスタであり、交流電流の制御には極めて好都合である。トライアックの点弧角はつまるところ、点弧パルス発生回路の電圧を調整することで変化させることができる。この電圧をポテンショメータの発生電圧 $u(t)$ (Answer Back Signal とも呼ぶ) とすれば、MDP により電球に加える交流電流を制御できる。

さて、直流モータは、加える電圧の極性で簡単に回転を逆転できる。また、電圧を加えなければ停止する。すなわち、高々2 ビットの情報があれば、モータの回転方向と停止を制御できる。我々の用いた AD ボード PCI-3170A は 16 点のアナログ入力とともに、2 点のディジタル出力が可能な構造である。これを用いて MDP の制御 (いわゆるポジションサーボ) を実施することができる。

具体的には、周期 T_c で変化する制御系からの操作量 u_k を T_c の間保持して (いわゆる 0 次保持) アナログ的には階段関数とし、今度はこの階段関数を設定値にして周期 T_s で On/Off 制御することにより、ポテンショメータの出力 $u(t)$ によりトライアックで負荷の電流を調整する。図 5 は、周期 T_s で動作している **TimeProc** の内部の MDP 制御系動作の動作構成図である。特に、ポジションサーボ系の On/Off 制御には、プラス/マイナス 5℃の不感帯を持つバックラッシュ機能を付加してある。また、図 6 は我々の製作したアクチュエータである。特に、内部の構成については、文字表示を加え説明してある。

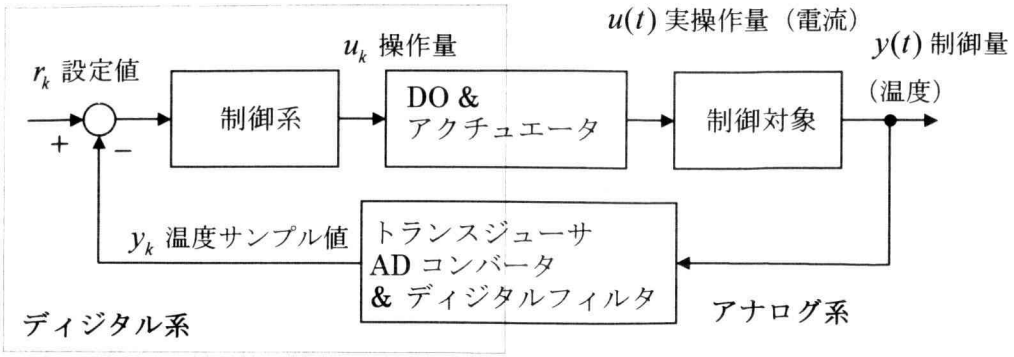

図 4　対象と制御系を含む全体系

6. プログラム構成の準備

制御系の本体はディジタルシステム、端的に言えばプログラムであり、更に具体的に言えば割り込み応答ルーチンである。2 節で既に述べたように、ハードタイマの割り込み応答ルーチン **TimeProc** とソフトタイマ Timer1 のイベン

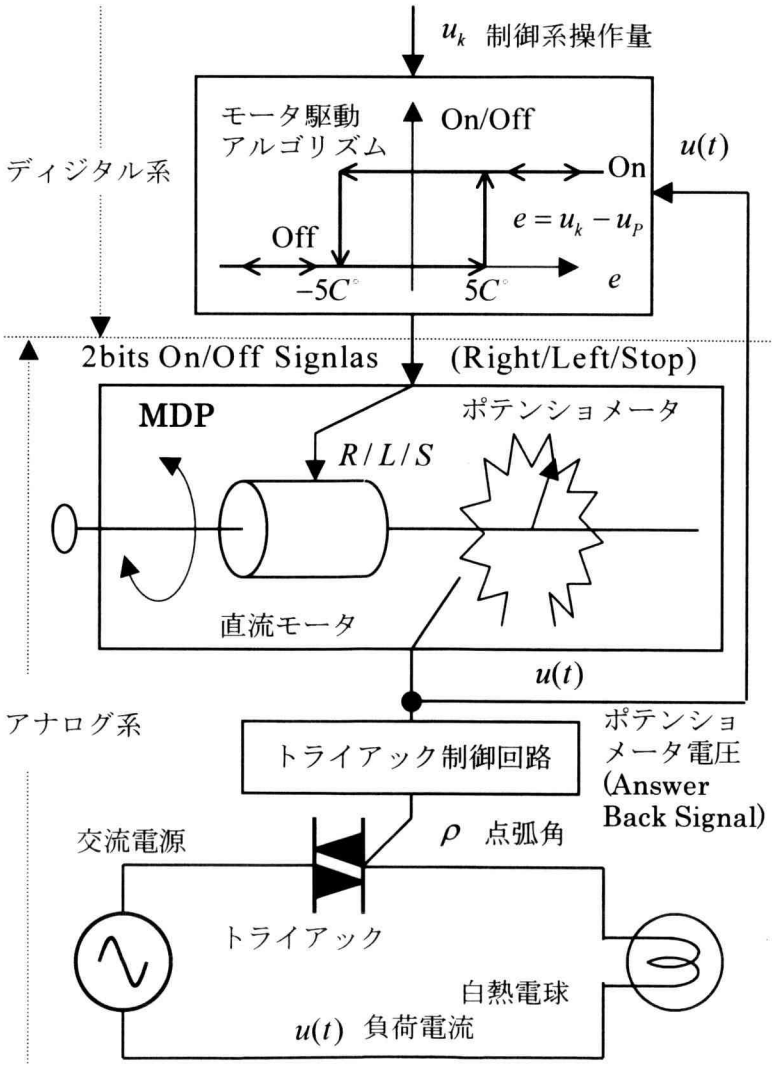

図 5 アクチュエータ構成図

トハンドラ **Timer1Timer** が我々の最終目的のプログラムルーチンである。

これらのルーチンを含み、各種の Visual なインターフェースを設定するために、プログラムシステム **BDDCont** を作成する。Borland の C++ Builder では、**BDDCont** プロジェクトと呼ぶ。この **BDDCont** には、通常の Visual 環境のプログラムには無い次のような特徴がある。

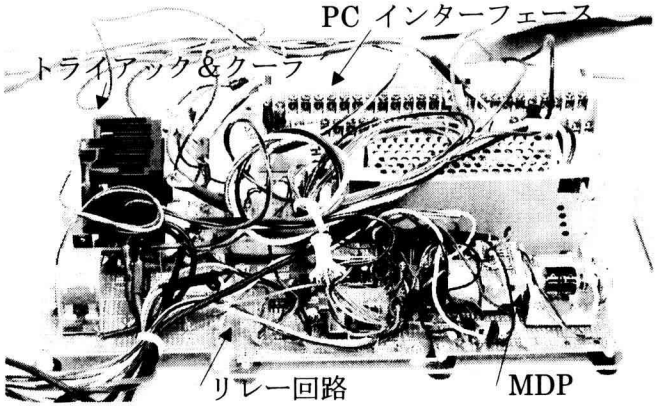

図 6　製作したアクチュエータ

(1) 通常のシステムでは使わないハードウエア（プロセス IO）を動作させる。
(2) プロセス IO を動作させるための関数パッケージを用いる。
(3) 時間割り込みの応答ルーチンを使用する。

第(1)の点では、プロセス IO を動作させるためのドライバが必要になる。これは、プロセス IO ボード（例えば、PCI-6103[1]と PCI-3170A[2]）に付属のドライバを手順に従ってインストールすればよい。

第(2)の点としては**ダイナミックリンクライブラリ**（dll ファイル）を使用する。これらの dll ファイルは先にドライバをインストールした際に、MS-Windows の Winnt¥system32 フォルダに書き込まれている。このダイナミックリンクライブラリィを使用するためには、更に、プログラムファイルの入っているフォルダに**ヘッダファイル**（h ファイル）を入れておく。ヘッダファイルは購入した際にボードに付属している。MS-Visual C++や MS-Visual Basic によりプログラムする場合には、これで準備は完了である。

しかし、Borland C++Builder を使用する場合には、プロジェクトフォルダへ**ライブラリィファイル**（lib ファイル）を入れておかなければならない。ライブラリィファイルはダイナミックリンクライブラリィファイルから、**implib.exe** プログラムにより作成する必要がある。ユーティリティプログラム **implib.exe** は、C++Builder の bin フォルダの中に含まれている。これを使用してライブラリィファイルを作る。例えば、bcFbiWtim.lib ファイルは、

implib　bcFbiWtim.lib　FbiWtim.dll

とすれば出来上がる。

さらに、ライブラリィファイルはプロジェクトに追加しておかなければならない。ファイルを付加するには、プロジェクトマネージャの **BTimerTst.exe** フォルダを選択して、マウスを右クリックすると現れるポップアップメニューの中の[追加]を選択して各ライブラリファイルを指定すれば良い。これらの関数の使用法詳細は、ボードに付属のヘルプファイルに記載されている。また、Internet で、Interface 社 URL: **http//www.interface.co.jp** を閲覧しても詳細なマニュアルを見ることができる。

7. BDDCont プログラム

前節で述べた**第(3)の点**は **BDDCont** プログラムの中で処理される。ここでは、極めて具体的に、C++Builder でプログラムを作成する手順に従って説明する。

まず、C++Builder を開き、[ファイル]メニューより[新規作成]を選び、その中の[アプリケーション]を選択する。これにより、Visual 環境のベースとなる Form1 という名前の Windows 上のシートが作られる。これで、TForm というC++Builder が持っているクラスライブラリィの中のクラスから派生した TForm1 というクラスが作られる。この定義ファイルは Unit1.h という名前のヘッダファイルに収められている。ここには **Form1** という TForm1 のインスタン

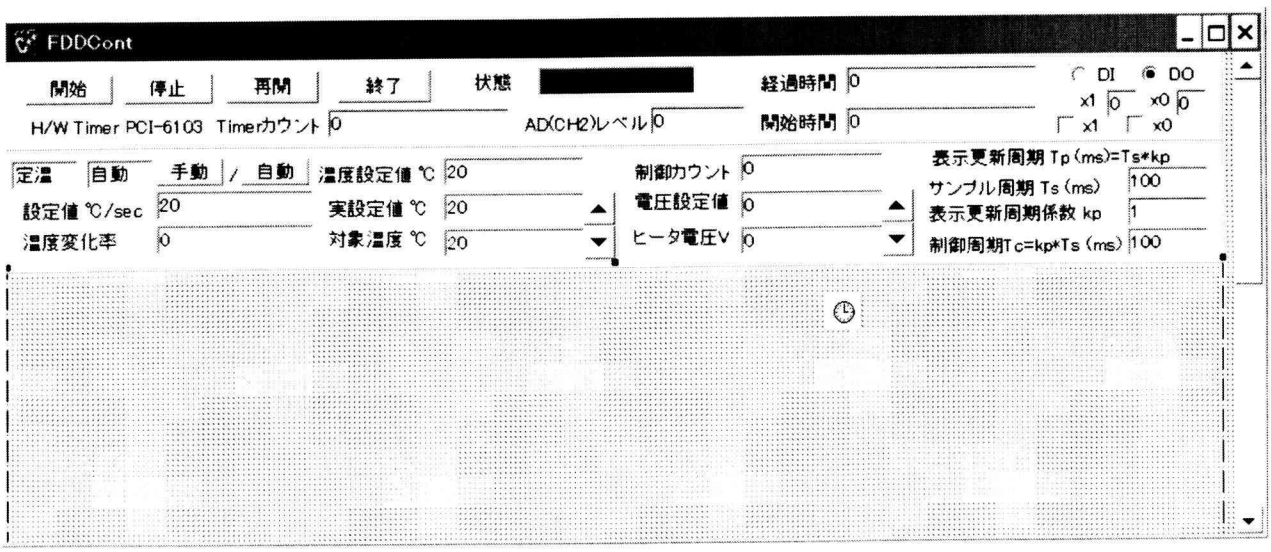

図 7　FDDCont フォーム上へのコンポーネントの配置

スが自動的にポインタの形で宣言される。**Form1** は TForm の性質を継承しているので、Windows 上のシートと対応している。これをずばり **Form1** と言って置けば、**Form1** 上に各種の Visual コンポーネントを貼り付けてゆくと、TForm1 のクラス定義の中にそれらのコンポーネントのインスタンスが挿入される（コンポーネントも夫々クラスになっている）。C++Builder では、押しボタン等のコンポーネントをダブルクリックすると、イベントハンドラすなわち、TForm のメンバ関数の定義の書かれる **Unit1.cpp** ファイルに自動的にリンクするようになっている。また、**project1.cpp** というファイルが自動的にできて、そこには、**Form1** を生成して（ポインタで定義されているので）、それを Windows 画面に表示する処理が書かれている。このプログラムについては、プログラム作成者は、終生これをいじる必要は無い。最初にプログラム作成者がやることは、これらの自動的に付されたファイル等の名前を変更することと保存することである。

具体的には、あらかじめシステムの中に **BDDCont** というフォルダを作成しておく。次に[ファイル]メニューより、[プロジェクトに名前を付けて保存]を選択する。すると、**Unit1.cpp** を保存することを問い合わせてくるから、まず、これを **BDDCont** フォルダに **BDDCont.cpp** というファイル名に変更して保存する。次にプロジェクトファイルの保存について問い合わせてくるから **BDDCont.bpr** として保存する。これで後は上書き保存してゆけばよい。次に、**Form1** の名前（プログラムの認識名あるいは、**Form1** のインスタンス名）を **FDDCont** に変更する。これには、**Form1** にカーソルを移し、そのプロパティ Name の値を **FDDCont** とする（以後、**Form1** ではなく、**FDDCont** と称する）。

さらに、**BDDCont** プロジェクトでは、**FDDCont** フォームの他に、ダイアログを使用する。ダイアログは一種のフォームであるが、モーダル処理などのために別に用意されている。具体的には、[ファイル]メニューより[新規作成]を選び[ダイアログ]タブの中の[標準ダイアログ]を選ぶ。これにより、**BDDCont** プロジェクトに **OKRightDlg** という名前のダイアログフォームと **Unit2.cpp** と **Unit2.h** というファイルが追加される。**Unit2.h** には TForm を継承した **TOKRightDlg** というクラスの定義が作られる。この中には既に 2 つの TButton クラスのインスタンス **OKBtn** と **CancelBtn** がポインタで宣言されている。これらのボタンをクリックした際のイベントハンドラすなわちクラスメンバ関数は **Unit2.cpp** ファイルにあり、自動的にリンクするようになっている。そこで、同じように、**Unit2.cpp** を **DDCont_d.cpp** と名前を付け替えて **BDDCont** フォルダへ保存する（これによりヘッダファイルも自動的に名前が変わる）。また、ダイアログフォーム名も Name プロパティを **OKRightDlg** から **DDCont_d** に変える。我々は、トレンドグラフや熱電対の変換プログラムなど汎用のプログラムを使っている。これらは、単純な形態のファンクションであるので、**UserProgram.cpp** というファイルにまとめてある。そこで、このファイルを **BDDCont** フォルダへ移し、**BDDCont** プロジェクトに追加してある。最終的なプロジェクトに関する重要ファイルの構成は C++Builder のプロジェクトマネージャに示される。

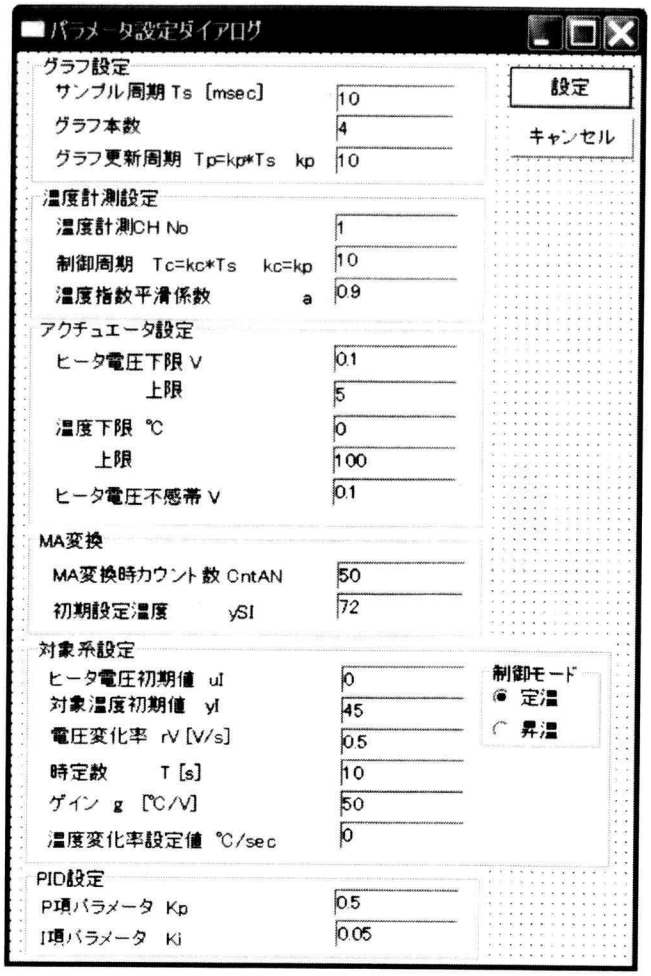

図 8 FDDCont_d ダイアログの構成

次に、PIO ボードの個性を収録したヘッダファイルをプログラムの中に挿入できるようにする。まず、自動的に生成されている **UserProgram.h** ファイルに、

```
#include "fbiwtim.h" // Timer 用ヘッダファイル
#include "FbiAd.h"   // AD 用ヘッダファイル
```

のようにヘッダファイルをインクルードしておかねばならない。そうして、**DDCont.cpp** と **DDCont_d.cpp** では **UserProgram.h** ファイルをインクルードしておかねばならない。

これに続いて、各フォームの設計を行う。メインフォームである **FDDCont** には図 7 のようにコンポーネントを配置する。また、ダイアローグフォーム **FDDCont_d** には図 8 のようにコンポーネントを貼り付ける。

フォームの設計が終われば、これらのコンポーネントの内、特にフォーム、ボタン等のコンポーネントをダブルクリックして、夫々のイベントハンドラを作成する。ここでは、メインフォームの開始ボタン Button1、終了ボタン Button5 と Timer1 のイベントハンドラ、および、ハードタイマによる外部割込み応答ルーチン **TimeProc** の 4 つについて説明する。

まず、Button1 のイベントハンドラの中で、ハードタイマ PCI-6103 と AI ボード PCI-3135 の初期化を行う。ライブラリィ関数 **AdOpen** により、AD ボードのハンドル

hDeviceHandle を取得する。また、TimerOpen 関数によりハードタイマのハンドル hHandle を取得する。以後、AD ボードは hDeviceHandle で、ハードタイマは hHandle で参照できる（**プログラム例 1** 参照）。

次に、AD ボードの読み込みのためのハードウエアの条件を設定する。ハードタイマは、周期の設定を行い、割り込み処理ルーチン TimeProc の名称を指定して使用可能状態とする。ハードタイマの実際の開始はライブラリィ関数 TimerStart を実行すればよい。これで、指定周期ごとに TimeProc 割り込み応答ルーチンが稼動し始める。ハードタイマ割り込み応答ルーチン TimeProc では、アナログ入力、アクチュエータ制御計算とディジタル出力、そしてソフトタイマの起動を行う（**プログラム例 2** 参照）。

次に、ソフトタイマ Timer1 のイベントハンドラ Timer1Timer の主要部は、図 2 のフローチャートに示したように、自らの動作を止め、トレンド表示を行い、制御系の計算を実施している（**プログラム例 3** 参照）。

最後に、終了ボタン Button5 のイベントハンドラ Button5Click を見てみよう。ハードタイマ、AI ボードの動作終了をライブラリィ関数 TimeClose と AdClose により実施している（**プログラム例 4** 参照）。

```
Void __fastcall TFDDCont::Button1Click(TObject *Sender) {
    //  ----------  途 中 省 略  ----------
    hDeviceHandle = AdOpen("FBIAD1");//AD 入力ボード初期化
    hHandle = TimerOpen("FBIWTIM", TIMER_FLAG_SHARE);
                                    // H/W Timer の初期化
    // AD ボード AD-3135 等 初期設定
    for (i=0;i<ADMaxCh;i++) {       // 例えば  ADMaxCh=3
        SmplChReq[i].ulChNo = i+1;
        SmplChReq[i].ulRange =ADRangeX(Gain);// ゲイン
    }
    // ---------- 途中省略 ----------
    // H/W Timer AD-6103 初期設定
    bSmpPrd=Pd->Period*1000; //H/W Timer 基本周期 Ts
    // タイマスタート
    TimerSetBaseClock(hHandle, bSmpPrd);
                    // Timer 基本周期設定 Ts[ms]
    TimerSetEvent(hHandle,nSmpPrd,TimeProc,
        (DWORD)hWnd,&dwTimerID,NULL,WM_NULL);
                    // Timer 周期倍率設定
    if (FDDCont_d->ShowModal()==mrOk) {
                    // ダイアログボックスモーダル表示
        TimerStart(hHandle, dwTimerID); // Timer スタート
    }
    // ---------- 途中省略 ----------
}
```

プログラム例 1　ハードタイマと AI ボードの開始準備

```
void CALLBACK TimeProc(DWORD dw, DWORD us) {
    // ---------- 途中省略 ----------
    // アナログ入力の読み込み処理
    nRet = AdInputAD(FDDCont->hDeviceHandle,
        FDDCont->ADMaxCh,AD_INPUT_SINGLE,
        &FDDCont->SmplChReq[0], FDDCont->pData );
                    // AD 入力処理
    // ---------- 途中省略 ----------
    FDDCont->syscnt++;  // システムカウンタ更新
    ua=(long double)FDDCont->pData[0];
                    // ポテンショメータ電圧  uP
    FDDCont->Cnt->u=(ua-scale)*5.0/scale;
                    // 制御出力(Answer Back 値)
    Data[0]=FDDCont->Cnt->u;
                    // -5V～5V or 0～5V 作図用
    y=(long double)FDDCont->pData[1];
                    // 温度(計測値)カウント値
    yv=-(y-scale)*5.0/scale;
                    // 電圧(mV) -5V～5V
    yin=KTCouple(0,yv); // mV->度  y
    FDDCont->Cnt->y=FDDCont->Cnt->a*FDDCont->Cnt->y
            +(1.0-FDDCont->Cnt->a)*yin;
                    // 指数平滑(Digital Filter)
    Data[1]=FDDCont->Cnt->y;        // 作図用
    // アクチュエータ制御
    uA=FDDCont->Cnt->uA;            // 制御系出力  uk
    if (FDDCont->Cnt->AutMan==0) {  // 0:自動 1:手動
        uS=uA;
    } else {                        // 手動
        uS=FDDCont->Cnt->uS;
    }
    FDDCont->Cnt->uSR=uS;// アクチュエータ設定値
    Data[2]=FDDCont->Cnt->uSR;      // 作図用
    Data[3]=FDDCont->Cnt->ySA;      // 温度設定値
    ue=FDDCont->Cnt->u-FDDCont->Cnt->uSR;// e=uk-uP
    if (fabsl(ue)<FDDCont->Cnt->h) { // Motor control
        ix=2;                        // off
    } else {
        if (ue>0) { ix=0;           // decrease
        } else {    ix=1;           // increase
        }
    }
    if (FDDCont->Cnt->uSR<FDDCont->Cnt->UL) {
        ix=3;                   // 電源停止
    }
    if (FDDCont->Cnt->u>FDDCont->Cnt->UH) {
        ix=2;                    // モータ停止
    }
    wDO=(WORD)ix;
    nRet = AdOutputDO( FDDCont->hDeviceHandle,wDO);
                                // ディジタル出力
    // ---------- 途中省略 ----------
    if (FDDCont->syscnt>0 && i==1) { // グラフ描画周期か
    // ---------- 途中省略 ----------
        FDDCont->Timer1->Enabled=true;// SoftTimer1 開始
        FDDCont->Timer1->Interval=100;// SoftTimer1 周期
    }
}
```

プログラム例 2　ハードタイマ割り込み応答ルーチン

```
void __fastcall TFDDCont::Timer1Timer(TObject *Sender) {
    // ---------- 途中省略 ----------
    Timer1->Enabled=false;  // Timer1 停止
    // ---------- 途中省略 ----------
    TrendCurv(Image1,Plt,Cnt,Pd->cntr); // トレンド描画
    //-------制御系------------------------------
    if (Cnt->CntA>0) {      // 手動→自動変換時
        yS=Cnt->ySA;
    } else {
        yS=Cnt->yS;
    }
    ey=yS-Cnt->y;           // 制御偏差  e=rk-yk
    if ((Cnt->u>=Cnt->UL) && (Cnt->u<=Cnt->UH)) {
        Cnt->euI+=ey;       // 偏差積分 wk=wk+e
        uA=Cnt->Cp*ey+Cnt->Ci*Cnt->euI; // PI 制御
        uk=p*e+q*w
    } else {
        uA=Cnt->uSR;        // 前回設定値
    }
    Cnt->uA=uA;
}
```

プログラム例 3　ソフトタイマのイベントハンドラ

```
void __fastcall TFDDCont::Button5Click(TObject *Sender) {
    // ---------- 途中省略 ----------
        TimerClose(hHandle);        // H/W Timer 終了
        AdClose(hDeviceHandle);     // AD 読み込み終了
    // ---------- 途中省略 ----------
    Close();
}
```

プログラム例 4　　終了ボタンのイベントハンドラ

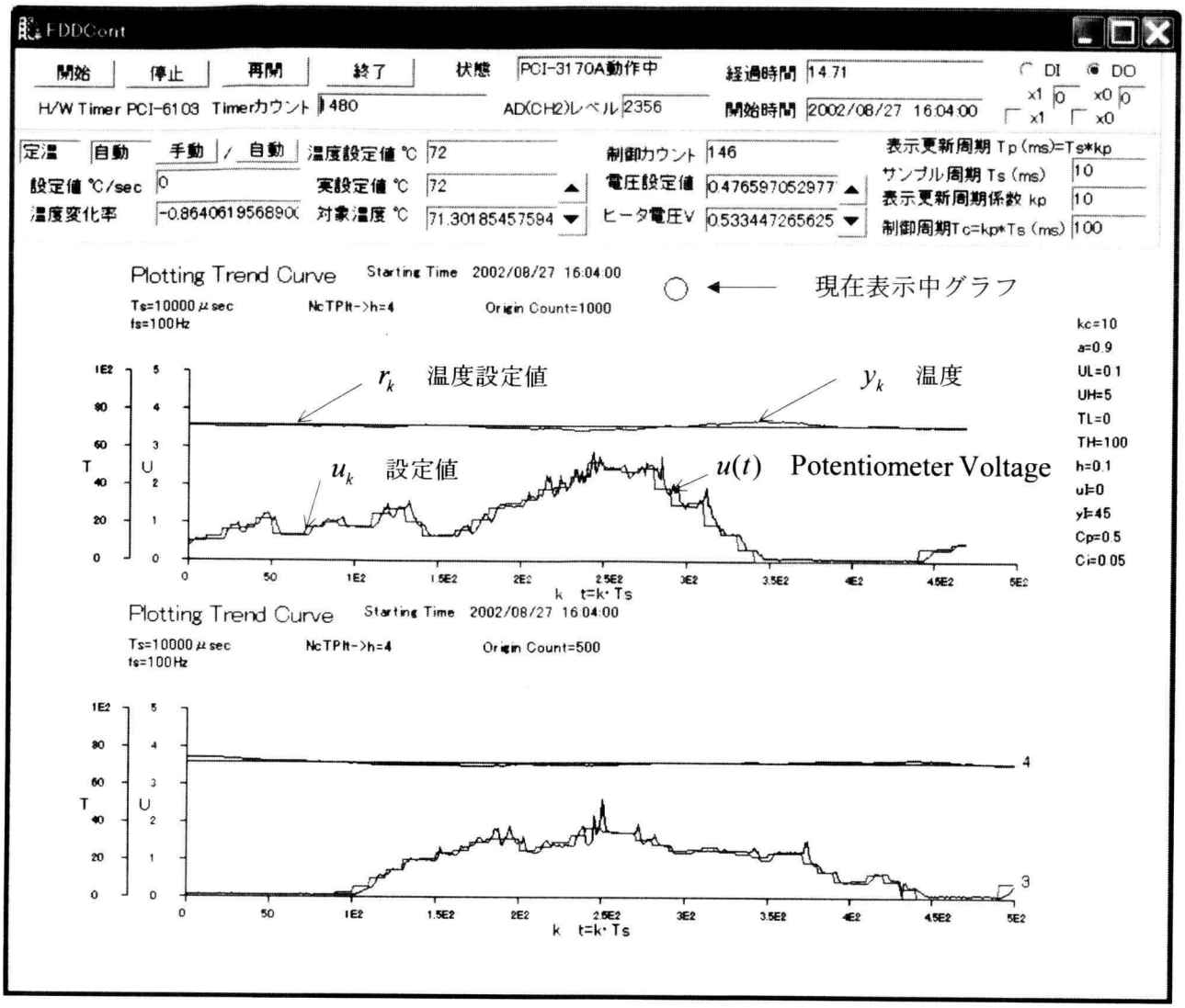

図 9 デフォルト設定での定温制御

8. 動作試験

作成したシステムを実際に稼動させてみる。図 9 は図 8 の設計画面でのデフォルト設定でのパラメータによる定温制御状態を示している。設定温度は 72 度である。信号サンプル周期はT_s =10msec であり、制御周期T_c = 100msec である。制御パラメータは $p = 0.5$, $q = 0.05$ の場合であるが、これは試行錯誤で変更できる。この設定が最適であるか否かは検討していない。制御周期は、実際に我々のシステム、すなわち、PentiumIV(1.6GHz)、MS-Windows XP 上では、T_s = 1msec, T_c =100msec でも、安定に動作することを確認している。これらパラメータの調整法は、学生の検討資料にはもってこいの題材となる。

図 10 では、一度手動に変更して電球に流れる電流を下げた状態にして温度を下げ、しばらくして、温度設定値 72 度のまま自動にしてみた例である。設定値は手動から自動に変更した時点での温度から目標設定値 72 度までの間を緩やかに直線的に実際の設定値を変化させ、温度を徐々に目標値に上げてゆくことができる。これがこのシステムのバンプレストランスファーの機能である。

9. まとめ

我々は、汎用のパソコンと汎用 OS の基で、ディジタル制御系構成例を与え、その作成と運用により有効な教育効果が得られることを示すことができた。すなわち、汎用パソコンの PCI スロットにタイマとアナログ入力用の 2 組のプロセス IO ボードを差し込み、リアルタイム環境を構成した。制御系は C++によりオブジェクト指向プログラムとして実現した。制御対象は白熱電球を熱源に用いた保温系で、アクチュエータとしては、MDP(Motor Driven Potentiometer)で駆動されるトライアック電流制御回路を用いた。このような構成により、制御系として有すべき、自動と手動との切り替え、特に手動から自動に切り替えた際のバンプレストランスファー、積分要素のアンチリセットワインドアップ、MDP を含む外部アナログ回路によるアクチュエータの持つフェールセーフ、そして制御アルゴリズムの機能の確認と検討の容易さ等を示すことができた。

今回製作提示したシステムは、PentiumIV(1.6GHz)、MS-Windows XP 上で、アナログ信号入力とディジタル出

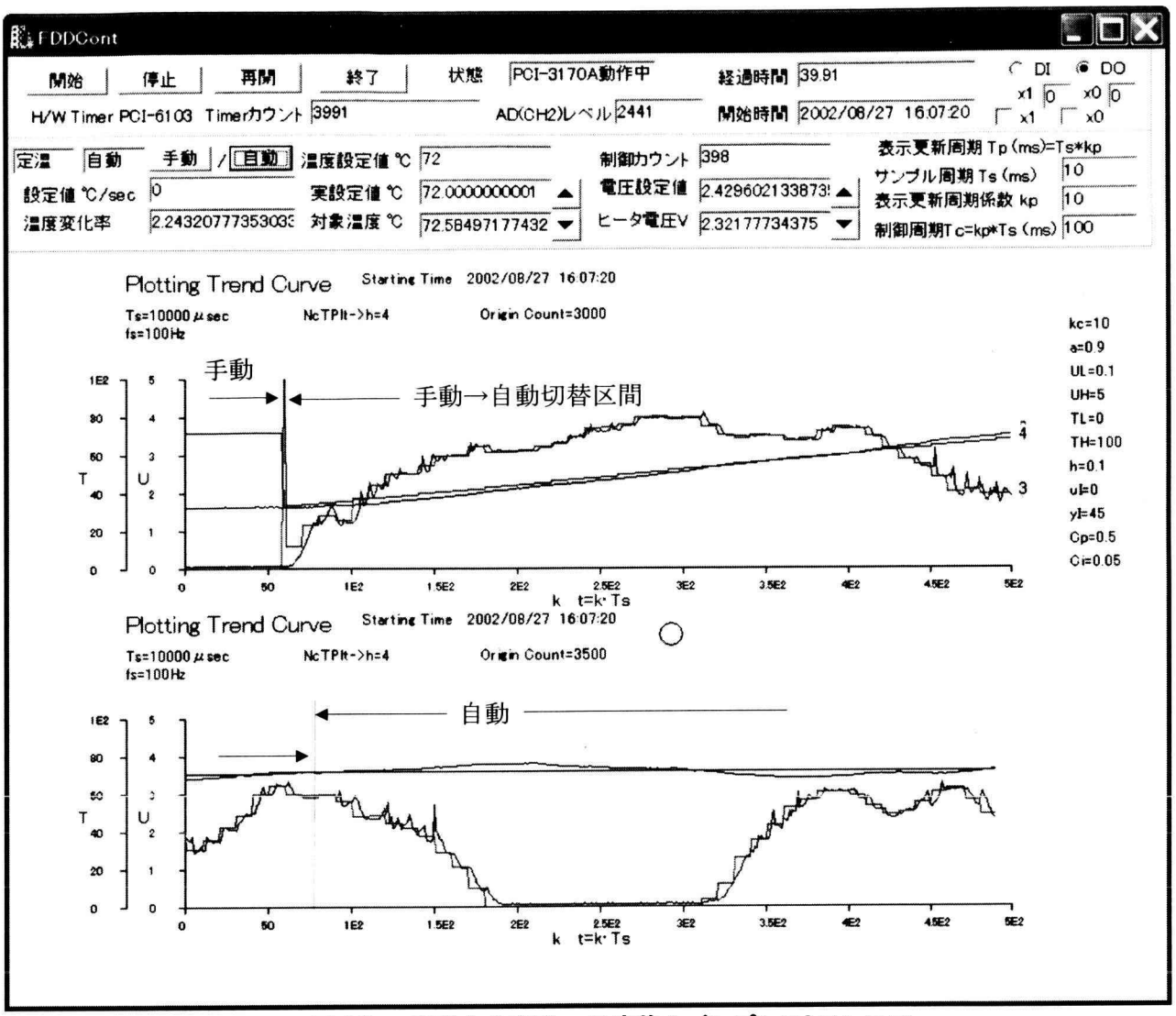

図 10　手動から自動への変換とバンプレストランスファ

力のサンプル周期として 10msec、制御周期として 100msec のシステムとして構成し、長期に渡り安定して動作できることを示すことができた。制御アルゴリズムは PI 制御で、試行錯誤によるパラメータ設定で動作させているが、必ずしも最適ではない。最適設定の問題は、今後、学生たちの課題として残すことができる。

卒業研究の対象として**ヒータによる容器内水の定温昇温制御系の高度化**を選んだ学生達は、この制御系を用いて検討し、プログラムを改装して機能向上を計り、あるいは、外部回路を拡張して容器の冷却制御回路を制作し、冷却ファンの動作シーケンスを制御する部分のプログラムを構成するなどして研究をまとめた。すなわち、このシステムは制御系理解のベースシステムとして有効な利用に供されている。

文献

[1] Interface「PCI-6103 ハードウエア USER'S MANUAL V1.5」、2000.11

[2] Interface「PCI-3170A ハードウエア USER'S MANUAL V1.0」、2000.10

[3] 大川善邦「Windows2000 による計測・制御プログラミングのノウハウ」日刊工業新聞社、2002.11

[4] Borland「C++Builder V6 開発者ガイド」、2002.3

[5] 白川洋充、竹垣盛一「リアルタイムシステムとその応用」朝倉書店、2001.9