卒業論文

オープン CNC マシニング・センタの高精度加工機能の研究

1p-　49p 完

平成 13 年 2 月 28 日提出

指導教官　　長尾 高明　教授

知能機械システム工学科
1010202　福見　寛重

目次

目次

第１章

序論

1 研究の背景と目的

日本の経済は 1950 年代、1960 年代の高度経済成長を経て目覚しい発展を遂げた。それを支えたのがコンピュータ技術の発達による大量生産の技術によるところが大きいと言える。従来、工作機械の操作は大部分が人手に頼っていたが、コンピュータによる数値制御(Numerical Control,NC)へと移行し、複雑な製品の加工を自動的に、繰り返し生産することが可能となった。また、複数の工程を要する製品の生産では、それぞれの工程での工作機械の自動化が進んでも、各工程において用いられる工作機械は異なる為、工程が進むたびに被加工物を工作機械に固定せねばならなかった。しかし、マシニング・センタのような 1 台で複数の工程を処理することのできる工作機械の登場により、このような問題も解決された。

これら一連の工作機械の進歩によって、大量生産システムが作り上げられた。その結果、生産現場において労働力が大幅に削減されたのである。

その後、日本が、世界の経済大国にとなるとともに、消費者のニーズが多様化し、ただ同じものをたくさん作れば良いという時代ではなくなった。人々はある程度の豊かさを得て、人とは違ったものを欲しがるようになったのである。それにより、近年言われるような「多品種少量生産」という生産モードへの移行が生産システムにも求められるようになった。

それまでの生産システムは、複雑であっても、同じものを生産するのであれば、大量に、そして速く、比較的高品質な製品を加工するという点では非常に優れていたが、多種多様な製品を効率よく、高品質な加工をすることはできなかった。そこで FMS(Frexible Manufacturing System)という混合生産ラインを念頭においた生産システムが考え出された。さらには CIM(Computor Integrated Manufacturing)というコンピュータ・ネットワークに基づく、製品の企画・開発から製造・販売に至るまでの各段階において発生する情報をすべて共有する集中管理の生産システムへと移行してきた。コンピュータ技術のさらなる進歩をとげた現在において、これらのシステムが生産効率、コスト、加工精度の向上に大きく貢献しているのみならず、生産量や労働者、流通システムなどの生産管理までが、コンピュータによって行われるようになり、これらのシステムの製造業に対する貢献は非常に大きいと言える。

しかしながら、現状の FMS や CIM では、種々の意志決定が企画や設計といういわば上流から、生産・販売といった下流へと一方向に流れていくトップダウン方式で行われている。従って，上流ではたくさんあった意志決定の選択肢が下流になるほどその選択の幅が狭くなっていくことで、しわ寄せを受け、そ

の結果，全体として硬直化してしまう傾向にある。

この問題を解決するには、上流での意志決定の際に下流での制約条件の情報を考慮する。すなわち、下流の情報を上流にフィードバックすることが必要である。

情報のフィードバックという点について、工作機械に関して言えば、人間が加工を行う場合に、五感や経験・知識に基づいて動作を決定していくように、現在の加工状態を認識したり、以前の経験を参照して加工を行う必要がある。

次世代では今まで以上に精度の高い製品が要求される。切削中、切削力や切削熱及び、びびり振動などの要因で、加工誤差は必ず生じる。しかし、現在、MC（マシニング・センタ）にはこのような製品を製作するために必要不可欠な加工誤差修正機能が備わっていない。そのため、今以上に高精度な製品を製作するのは困難である。この加工誤差修正機能が実現されれば、容易に高精度な製品が製作できる。

本研究の大きな目標はマシニング・センタに誤差修正機能を付けることである。この目標に向かい、三つの部門に分けて研究を進めていくことにした。まず一つ目の研究はマシニング・センタの加工中の力をセンサで測定し、コンピュータ画面中にリアルタイムで表示する。二つ目の研究は一つ目の研究で測定したデータを元にしたデータベースの作成。三つ目が私の研究で、この二つの研究結果からのデータを元にして、マシニング・センタをリアルタイムで操作し、誤差の修正をする。

第２章

精密・知能化加工の概念

2.1　精密・知能化加工

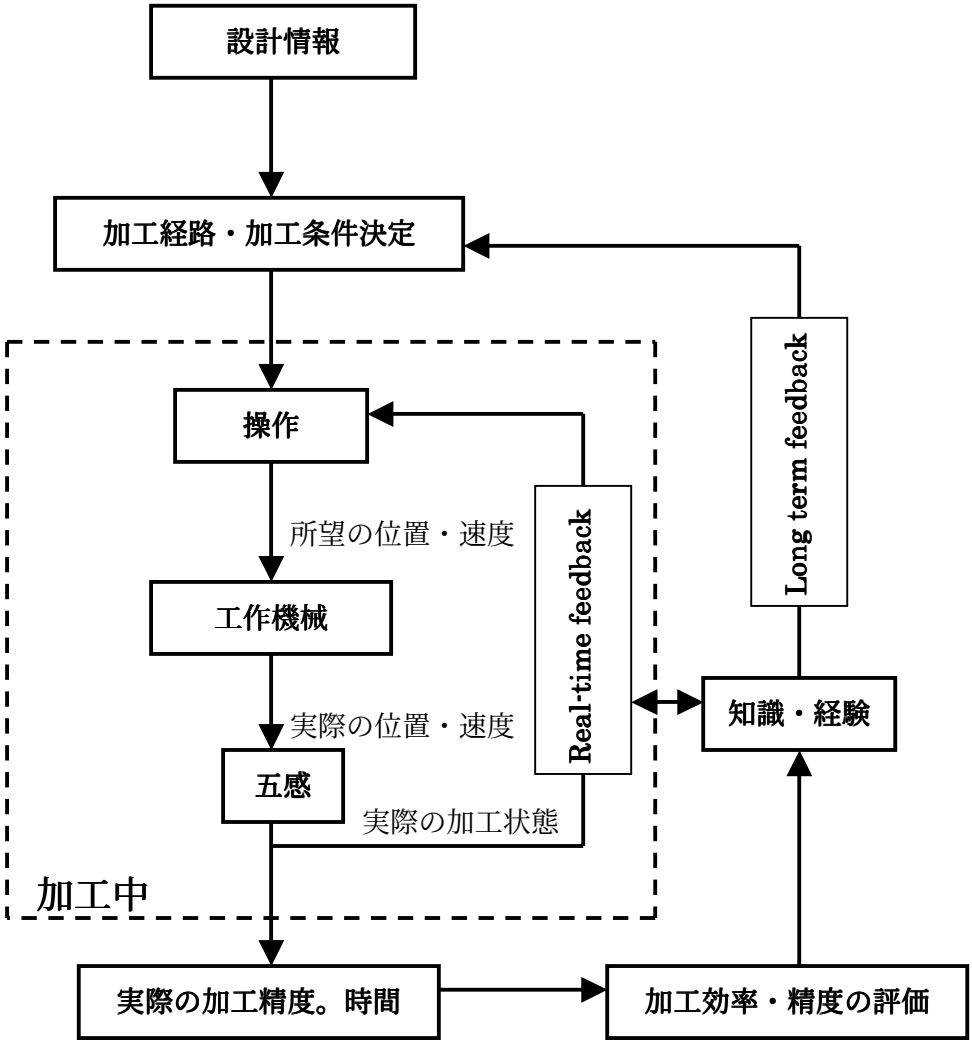


図 2.1　熟練加工者の製作工程

図 2.1 に示すように、熟練加工者は加工を行うことによって、加工中の切削音、臭い、振動などの現象を目や耳、鼻などの五感でとらえ、それらと加工の結果で得られた仕上がり、加工条件の関係を知識や経験として蓄えている。それらを次回以降の加工工程の決定に役立てており、この点でサイクルタイムの大きいフィードバック（図中の Long term feedback）を形成しているといえる。

従って熟練加工者が品物を製作する過程は二重のフィードバック系によって構成されていることになる。

本研究の目的は、短時間で精度の良い加工を加工の知識や経験がなくとも実現する知能化加工システムの構築である。そのためには、熟練加工者には認識できる加工中に起こっている物理現象を工作機械からの情報で認識して、制御に

反映させたり、あるいは次回以降の加工に役立てていくという作業を、工作機械の簡単な操作でできるようにする必要がある。

そこで前述の熟練加工者と工作機械を比較して考えると加工中にはリアルタイムに加工状態を測定、把握することで、それに応じた制御指令を工作機械に送り、なおかつ、加工中に学習したことをデータ・ベースへと反映させていくことで次回以降の加工条件の設定やリアルタイムでの制御などに役立てていく必要がある。いずれの場合にも必要なことは、加工中に生じる物理現象を工作機械を操作する人間がが認識することが不可欠である。すなわち、

1. センサ融合型加工機械が自らの状態をリアルタイムで表示し，判定結果を元にし利用者がこれを制御する。
2. 加工中にセンシングしたデータにより、加工条件と加工状態に関するデータ・ベースを蓄積、保有し、それを参照することで、加工条件の設定や制御を行う。

という機能が必要なのであって、そのためには熟練加工者の五感に相当するものが不可欠となるのである。従って、熟練加工者の五感に相当するものとして、加工機械に種々のセンサを搭載し、そのセンサでの情報を解析する機能、そして、その情報を元にして正確にコントロールできるコントロールパネルが必要となる。

従って本研究のシステムはオープン CNC マシニング・センタの高精度加工システムとする。

2.2　ネットワーク対応型加工システムの意義

センサ情報に基づく知能化加工システムでは、

・工作機械がセンサ情報によって得た情報をコントローラに返す

・コントローラが工作機械に対して制御指令を送る

・蓄積された加工の知識を実時間コントローラが参照する

といった各要素間において、互いに頻繁なアクセスがある。

そのためには、各要素間が相互に緊密な連絡を取り合えるような、ネットワークが必要となる。

2.2.1　自立分散型ネットワーク

システムの統合を基本概念として、コンピュータネットワークを用いた生産システムとして CIM がある。

CIM では、設計・生産の過程では CAD（Computer Aided Design）を用いて製品の設計が行われ、CAM（Computer Aided Manufacturing）を用いて設計情報から加工情報への変換が行われる。CAPP（Computer Aided Process Planning）は、部品形状から加工の手順、工程の設計を行うものであり、最近ではこれらに加えて、統計解析の手法として CAE（Computer Aided Engineering）も用いられる。

これら設計・生産の過程は当然として、営業や管理部門、さらには部品の供給先やユーザまで含めて、生産に関する広範囲な対象の多種多様なすべての情報を統合し、変化する生産環境への対応を目指すものである。システム全対象にまたがる情報ネットワークの構築を具体化するための方針として、集中管理型のネットワークを採用し、製造業界において大きな関心を集め、成果を生み出しているところもある。

システム全体で各種の情報を共有できるということは大きな利点だが、集中管理型であるがゆえの問題点がある。

大量生産の製品が設計まで終わって、あとは生産・販売を繰り返すという場合には、専用の生産システムを構築し、一定の条件で生産を続ければ良く、集中管理型のシステムが非常に有効である。しかし、消費者の多様なニーズに答えるような多品種少量生産のシステムや製品の設計過程などでの試作の作成となると、対象によって生産する際の条件の設定が異なる。また、対象によって生産ラインを構築し直したり、システムに要素を付け加える必要がある場合などには、従来のシステムではその影響が全体に波及してしまうため、一部を変更するだけではすまない。このような点で現状の CIM はフレキシビリティに欠ける。

また、研究機関での実験装置の作成や、製造業界における開発段階での設計―試作―設計―試作といった毎回扱う商品が違うような環境では、なおさら加工システムに要求されるフレキシビリティが増すが、現状の CIM のような集中管理型のシステムは的確ではない。

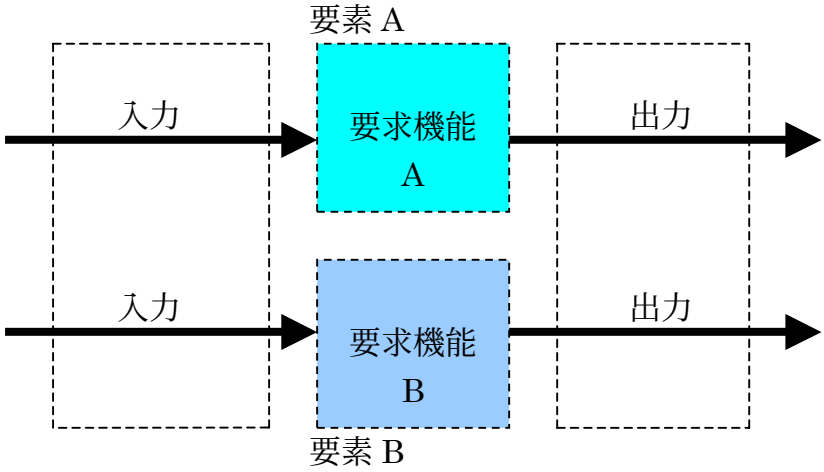


図 2.2　自立分散型の要素

そこで、これからの問題の解決策として,自立分散型のシステムが考えられる。自立分散型システムとは、システムを構成する要素（加工システムであれば、CAD、コントローラ、データ・ベースなど）が結びついたネットワークを構成することが可能である。しかしながら、それぞれの要素他の要素とは独立で機能することができ、得たい機能にあった要素に対して必要な情報を決まった形式で入力さえすれば各要素が内部で自立的に機能し、出力を吐き出すというシステムである（図 2.2)。

特に自立分散型システムの特徴としてあげられることは、要素同士の親和性が良く、どの要素とでも組み合わせることが可能であること。また、希望する機能によって要素の置き換え（図 2.3)、追加、削除が可能なことである。つまり、多種多様なものを生産する過程で、新たな機能が欲しくなった時や、機能の交換が必要になった時にその部分だけ変えれば良いのである。

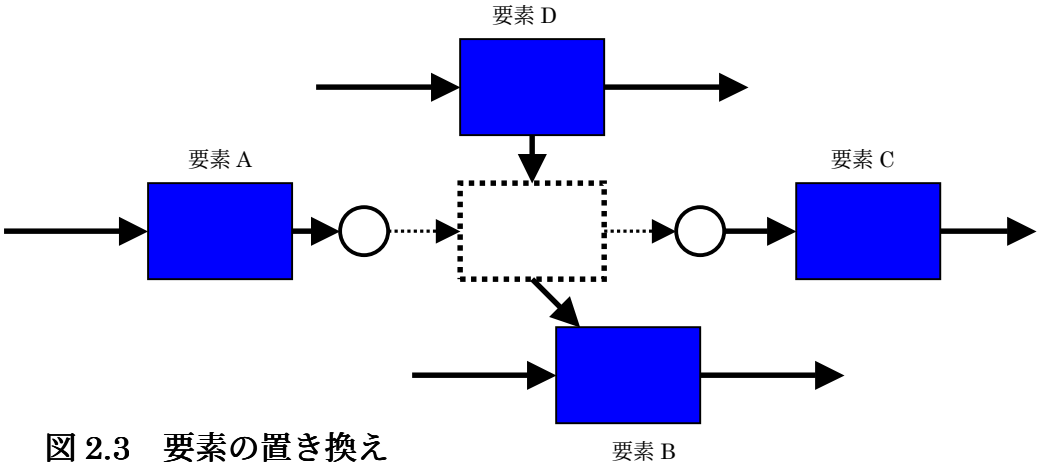


図 2.3　要素の置き換え

2.2.2　コンピュータネットワークの利用

知能化加工システムが

・センサ情報のフィードバックに代表される各要素間での情報の共有を実現しながらも、

・試作加工などに適した十分なフレキシビリティを持つためには、自立分散性を持ったネットワークが必要であることは前節までに述べた。

システムが自立分散性を有するためにネットワークとして必要な機能の一つとして、ネットワークを構成する上での様々な規則（プロトコル）が確立されていて、それに従った典型的な入出力が可能なことがある。

また、各工程で情報を共有し、相互間でのアクセス可能なシステムにおいて、ネットワーク機能に要求されることとしては、工作機械の制御情報だけでなく、センサ情報や加工の知識などの多種多様な情報を取り扱える汎用性もあげられる。

これらの機能を果たすものとしては、普及したコンピュータネットワークが考えられる。以上のようなことより、本研究ではコンピュータネットワーク上で加工システムを構築する。

加工システムをコンピュータネットワーク上に構築することは、

・世界的に普及しつつあり、多様な情報へのアクセスが可能である。従って、企業や大学の持つ各種のデータベースへのアクセスが可能である。

・普及しているために比較的安価である。

・ネットワークを通じて、遠隔地のコントローラからのアクセスも可能である。

・パケットの操作などの通信プロトコルが確立されているために、入出力形式が決まっていて,システムの要素の中身をあまり意識しなくても、システムの構築ができる。したがって、システムの構築が比較的容易である。

・工作機械でもネットワーク上の一台のワークステーションのように扱える。工作機械に慣れない人間にとってはそれだけでもユーザインターフェースの面で優れていると言える。

また、2.2.1 でも述べたが、

・新たな機能が欲しくなった時や、機能の交換が必要になった時にその部分だけ変えれば良い。

以上のような利点がある。

また、試作加工という点から見れば、現在では一般的に製品の設計は EWS などのコンピュータ上で CAD/CAM システムを用いて行われ、その情報を試作用の工作機械に渡して加工している。

このようなことを考えれば、コンピュータとコントローラ、工作機械が同じネットワーク上に存在し、コントローラが CAD/CAM で発生した加工情報を用いて、ネットワーク上で加工機械を制御できることは、設計段階のものをすぐにしたいという要求に対して、大変有意義である。

一般的に設計者は CAD を日常的に使い、コンピュータには慣れているが、工作機械には不慣れであることが多いため,、そういう意味でもユーザインタフェースに優れていると言える。

第３章

実験装置及び実験方法の概要

3.1　実験装置の概略

3.1.1　オープンアーキテクチャ CNC マシニング･センタ

図 3.1　オープンアーキテクチャ CNC マシニング･センタ

オープンアーキテクチャ CNC マシニング･センタ（MC）
・HSSB（High Speed Serial Buss）を経由して外部パソコンから制御できるオープン CNC を搭載した OKK　VM4Ⅱマシニング･センタ

3.1.2　PCPG46

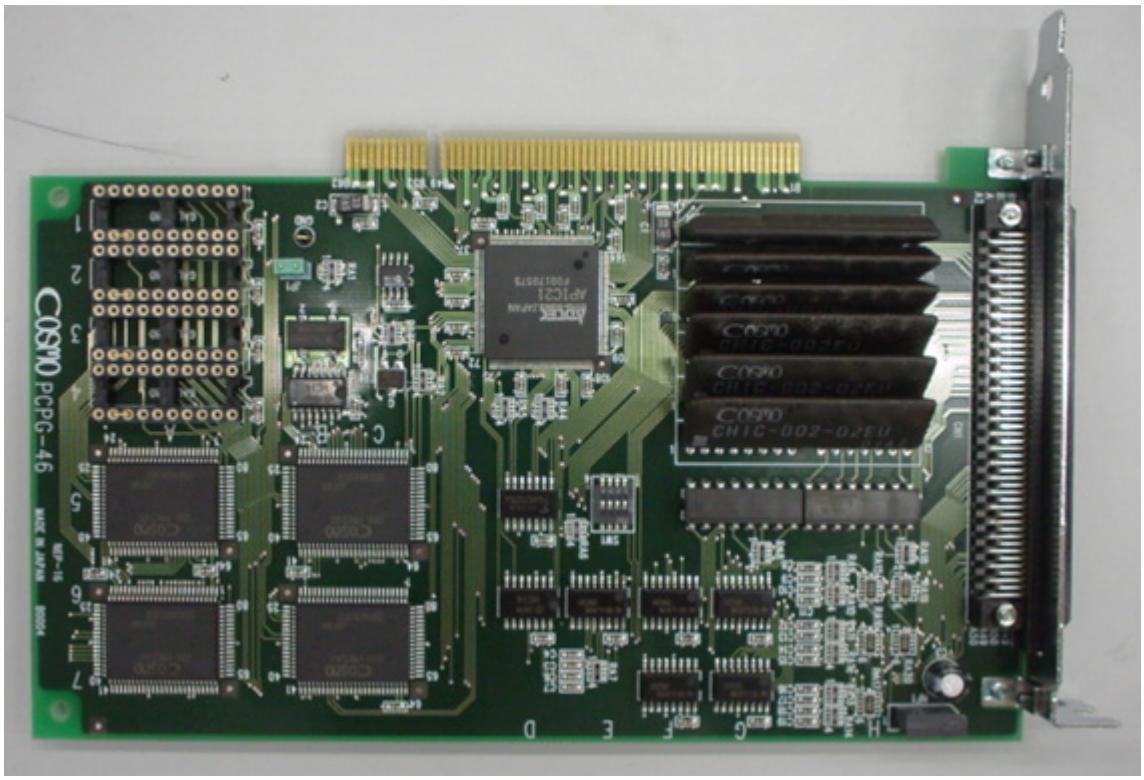


図 3.2　PCPG46

PCPG46

専有アドレス	64アドレス使用
外形寸法	106.68×174.63mm（コネクタ部、パネル部含まず）
入力電源	DC＋5V±0.25V
消費電流	Min 200mA Max 1000mA
制御機能	PRESET PULSE DRIVE （指定パルス数ドライブ） CONTINUOUS DRIVE （連続ドライブ） SIGNAL SEARCH-1, 2 DRIVE （信号検出ドライブ） 最高出力周波数…4.096MPPS 速度オーバーライド機能 移動量オーバーライド機能 直線加減速機能 自動S字加減速機能 出力パルス数管理用28Bit Counter内蔵（各軸毎） フィードバックパルス数管理用28Bit Counter内蔵（各軸毎） 偏差量自動算出機能 急停止機能（入力信号、コマンド書き込み 何れも可能） 減速停止機能（コマンド書き込み） リミット停止機能（急停止、減速停止 何れも可能） 最大4軸制御（同時ドライブ可能）
出力信号	パルス出力等…ラインドライバによる差動出力 ドライバ制御出力…フォトカプラによるオープンコレクタ出力
入力信号	機械系入力及び汎用入力…＋12～24V フォトカプラ入力 ドライバステータス入力…＋12～24V フォトカプラ入力 フィードバックパルス入力…ラインレシーバーによる差動入力
使用可能ドライバ	ステッピング及びサーボモータドライバ パルス列入力型（1パルス/2パルス 何れも可能） 差動入力、フォトカプラ入力、TTL入力型

3.1.3　PG46－CNV

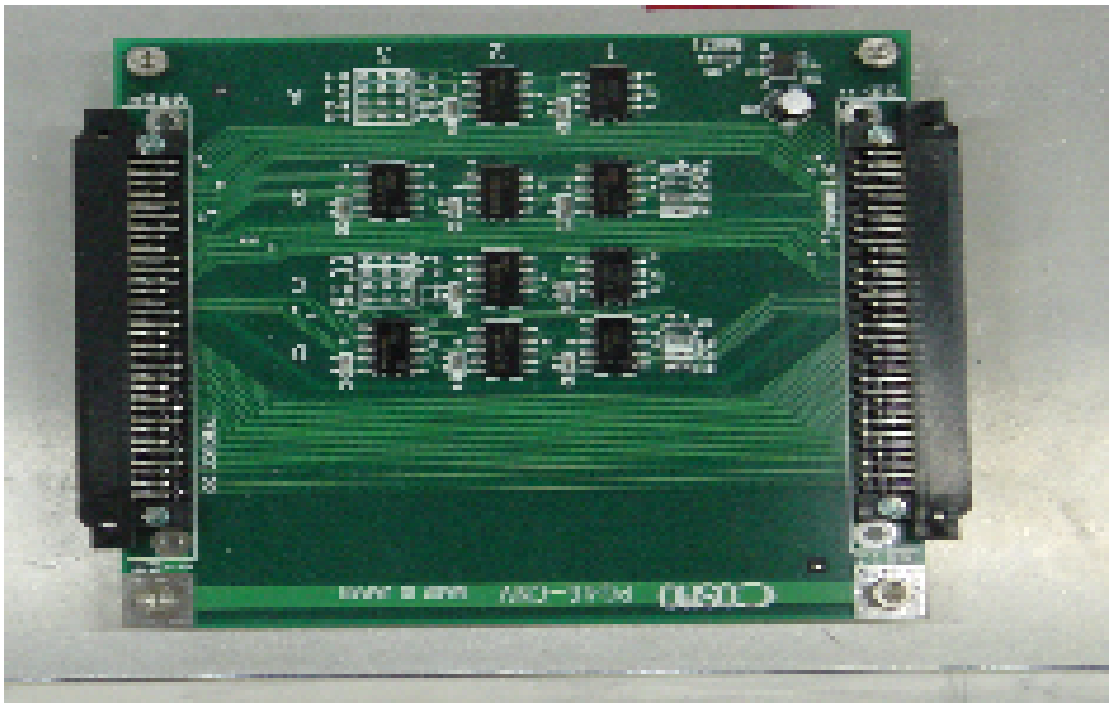


図 3.3　PG46－CNV

PG46－CNV
コスモテックス社製モータコントロールボードのパルス出力信号（DIR/PULSE）を、2 信号（A 相/B 相）に変換するための中継ボードであり、1 ボードで最大 4 軸分の変換が可能である。

3.1.4　PCPG46 ボードを搭載した操作用コンピュータ

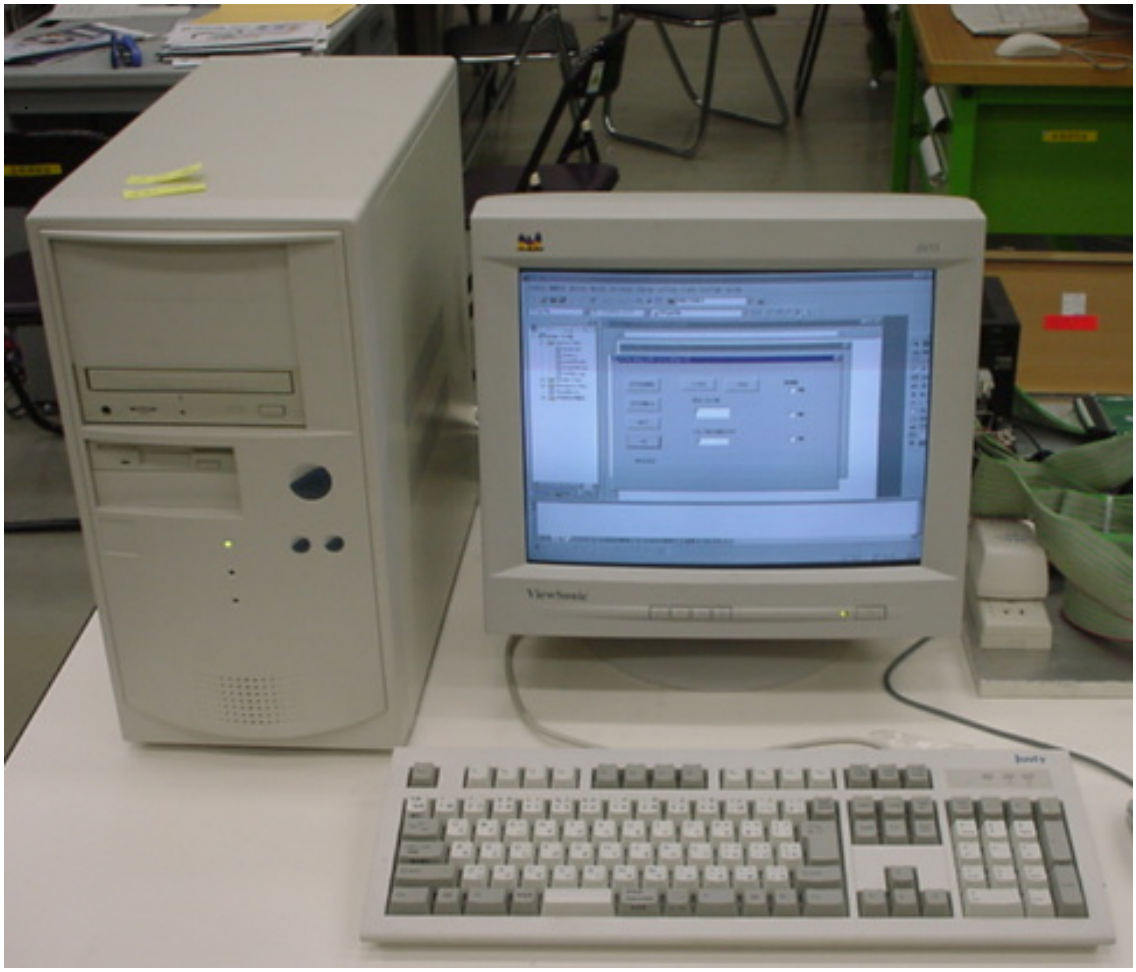


図 3.4　操作用コンピュータ（1）

3.1.5　HSSB ボードを搭載した操作用コンピュータ

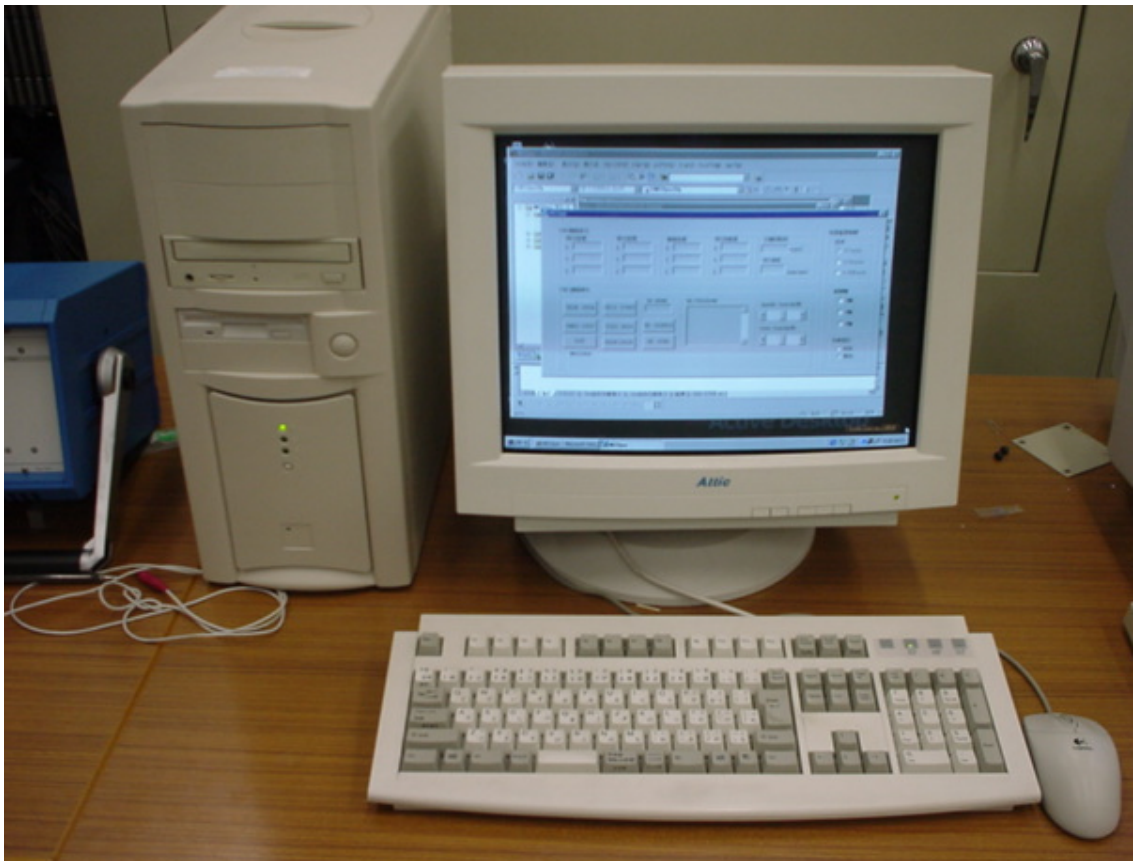


図 3.5　操作用コンピュータ（2）

3.1.6　実験装置の概略

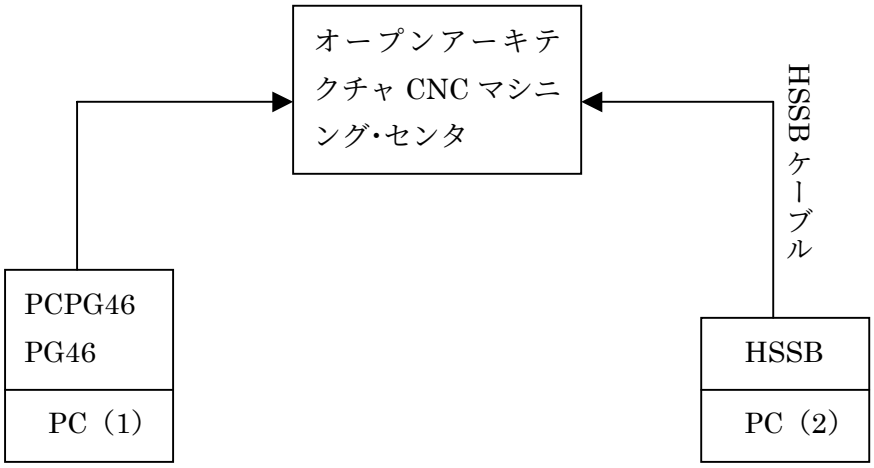


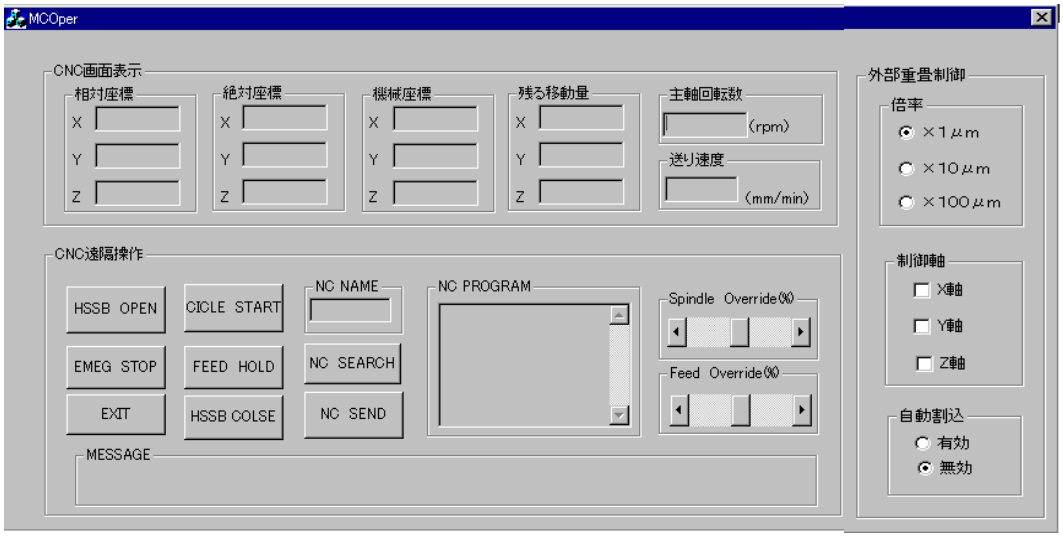
図 3.6　実験装置の概略図

3.2　実験の概要

・VisualC++を使用して、マシニング･センタの操作画面及び、割り込み操作を行う画面を作成する。
・マシニング･センタは 1 パルスで 1μm 移動すると OKK のマニュアルには記載されていたが、それを確認する。
・VisualC++で作成したコントロール画面を使用して、実行中（切削中）のマシニング･センタの NC プログラムに割り込み、制御する。

3.3　コントロール画面の概略

PC（1）コントロール画面【1】
この画面で割り込みのプログラムを作成する。
発生パルス数の欄に数字を入力し、＋方向か－方向のボタンを押すとパルスが発生し、発生させたパルスの分だけマシニング・センタが移動し始める。また、制御したい軸のチェックボックスをチェックするとその軸の制御ができる。複数の軸を制御したい場合には複数のチェックボックスをチェックすることでその制御ができる。



PC（2）コントロール画面【2】
この画面から MC の遠隔操作を行ったり、切削状況のリアルタイムでの表示や

割り込み制御の有効、無効を決定する。また移動距離の倍率を変化させることにより、移動距離も変化する。先ほどのコントロール画面【1】と同じように制御したい軸を選びチェックする。この時、操作画面【1】と【2】とが同じ欄にチェックされていないと制御ができない。割り込み制御をおこなうには自動割り込みのチェックボックスを有効の欄にチェックする必要がある。

第４章

実験

4.1　移動距離の確認

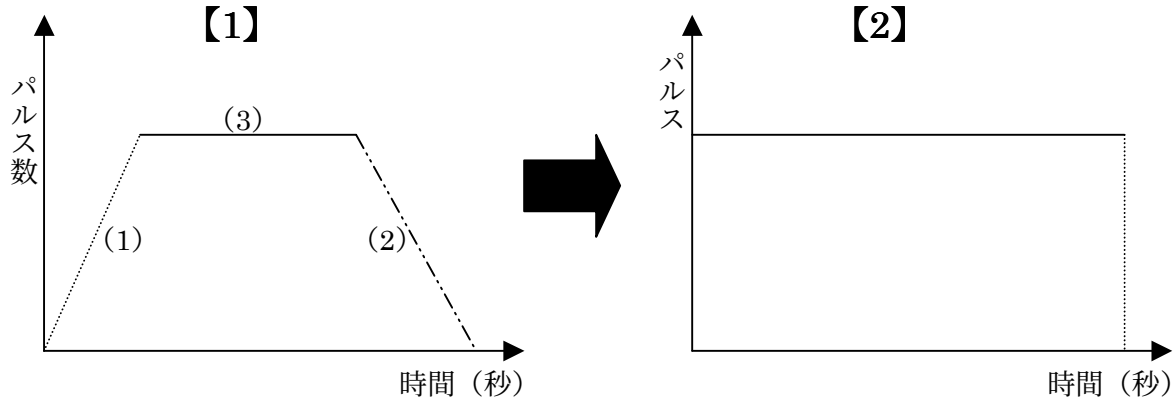


図 4.1　パルスの発生の仕方

先に述べた通り、OKK のマニュアルには 1 パルスあたり、1μm 移動すると記載されていたが、コスモテックス社製の操作画面を使って調べた結果、1 パルスで 0.25μm 移動することが分かった。

コスモテックス社製の操作画面は【1】のようなパルスの発生のさせかたをしていた。しかし、この発生のさせ方では 1 パルスあたりの移動距離を割り出すことはできなかった。その理由は（1）と（2）と（3）で発生したパルス数を足して、その数で移動距離を割ることで 1 パルスあたりの移動距離を求めることができるが、パルスは最大の時【(3）の時】に安定するため、(1）と（2）の時は不安定で移動距離も安定しなかった。

そこで、設定を変更して、【2】のように始めからパルスは最大に発生させた。そうすると移動距離も安定した。さらに、どのような条件で実験しても 1 パルスあたり、1μm 移動することが分かった。

4.2　実験方法

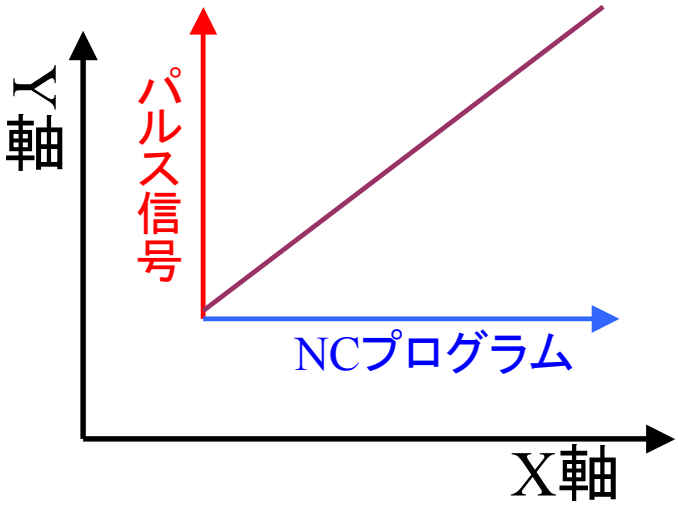


図 4.2　実験結果の予想図

S3000　　M03；
G91　　G01　　X90.0　　F180；
M05　　M02；
この NC プログラムは X 軸方向に 180 ㎜/sec の送り速度で、90 ㎜切削するというプログラムである（このプログラムは 30 秒間で終了する）。

　このような NC プログラムに 4000PPS を 30 秒間連続的に発生させて、合計で 120,000 パルスを発生させるプログラムを割り込ませる。このパルス信号は 1 秒間に 1mm ずつ、Y 軸方向に 30 秒間移動させるものである。

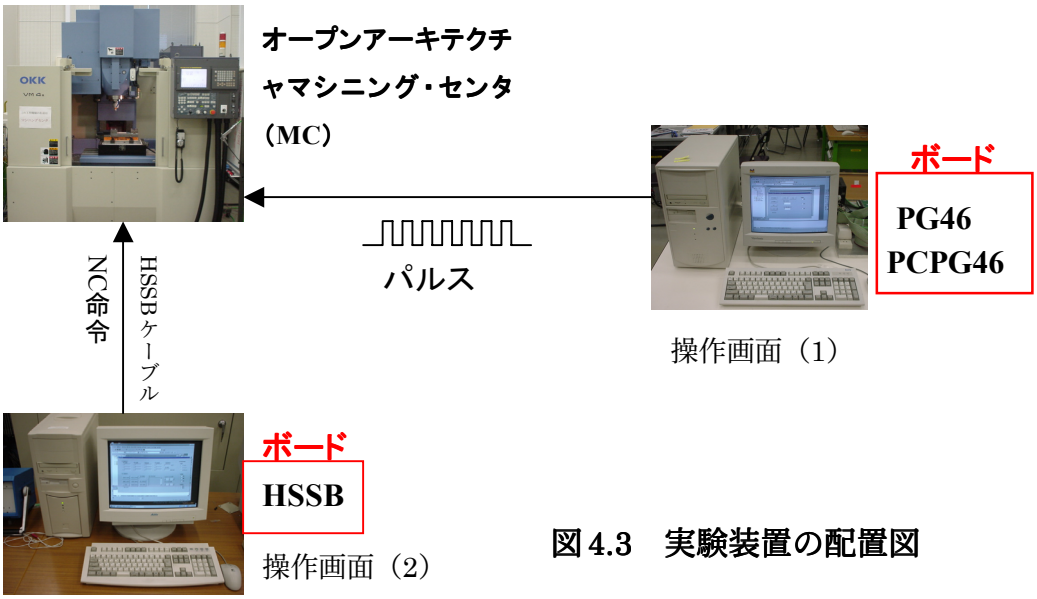


図 4.3　実験装置の配置図

4.3　実験結果

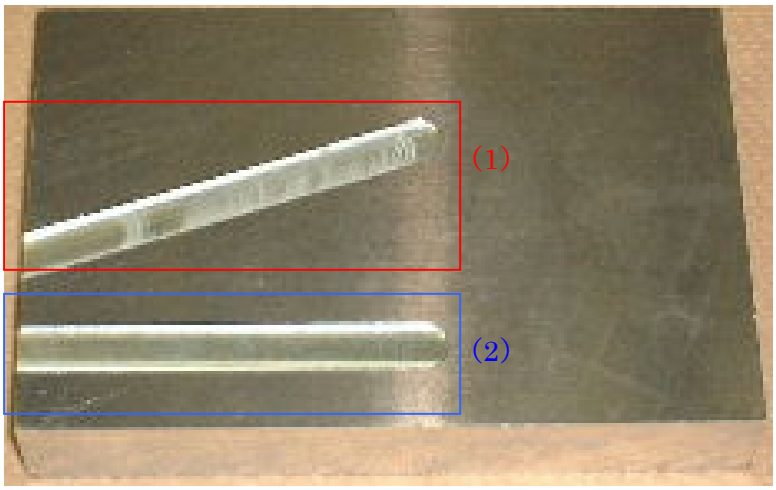


図 4.4　加工結果

・(2)の部分がマシニング・センタのNCプログラムで、送り速度180(mm/min)でX軸方向に30秒間移動させたものである。
・(1)が(2)のプログラムに1mm/secでY軸方向に移動するパルスを割り込ませたものである。

(1)と(2)、2 つの切削跡の終了点を測定すると、(1)の切削跡の終了点は (2) の切削跡の終了点より Y 軸方向に 30 ㎜移動していた。

4.4　まとめと問題点

加工中のプログラムに割り込み制御することはできるが、まだ複雑な動きができないのでこれを改善していく必要がある。また、誤差の修正をするには、他の研究と組み合わすことが必要である。この課題を解決することにより、加工誤差修正機能は実現可能な装置である。

この研究の問題の一つとして、NCプログラムによる切削方向と全く逆方向に移動するパルス信号を入力すると、理論的にはマシニング･センタの動きはストップしてしまうはずであるが、そのパルス信号は MC に認識されず、NC プログラムだけが実行された。

操作画面（1）を MC と接続すると、MC 自体の外部制御の有効、無効や、コントロール画面での自動割り込みの有効、無効に関係なく MC 自体でのハンドル操作ができなくなる。これは MC の内部プログラムの問題なので、操作画面（1）との接続を解除するか、製造元である OKK に改善してもらう。

謝辞

本研究を進めるにあたり、多くの方々に御指導、御協力いただきました。

指導教官である長尾教授、李軍旗助教授には研究全般から日常生活の多岐にわたり、並々ならぬ御指導をいただき有難う御座いました。

同じ研究室の木崎さん、更谷さん、増吉さん、田中、大谷、現田、柳原には大変御世話になりました。これからも友達でいてください。

以上の方々に深く感謝し、厚くお礼申し上げます。

付録

```
// pcpgDlg.cpp : インプリメンテーション ファイル
//

#include "stdafx.h"
#include "pcpg.h"
#include "pcpgDlg.h"
#include "pcpg46.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
int A=0;
#endif

pcpg46 pcpg;

TCHAR                   szBuf[1024];
BYTE                    bData[256];
WORD                    wBsn, wAxis;
PCPG46RESOURCE   ri;
BYTE                    b;
DWORD                   dwCount;

/////////////////////////////////////////////////////////////////////////////
// アプリケーションのバージョン情報で使われている CAboutDlg ダイアログ

class CAboutDlg : public CDialog
{
public:
CAboutDlg();
```

```
// ダイアログ データ
//{{AFX_DATA(CAboutDlg)
enum { IDD = IDD_ABOUTBOX };
//}}AFX_DATA

// ClassWizard は仮想関数のオーバーライドを生成します
//{{AFX_VIRTUAL(CAboutDlg)
protected:
virtual void DoDataExchange(CDataExchange* pDX);    //
DDX/DDV のサポート
//}}AFX_VIRTUAL

// インプリメンテーション
protected:
//{{AFX_MSG(CAboutDlg)
//}}AFX_MSG
DECLARE_MESSAGE_MAP()
};

CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)
{
//{{AFX_DATA_INIT(CAboutDlg)
//}}AFX_DATA_INIT
}

void CAboutDlg::DoDataExchange(CDataExchange* pDX)
{
CDialog::DoDataExchange(pDX);
//{{AFX_DATA_MAP(CAboutDlg)
//}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)
//{{AFX_MSG_MAP(CAboutDlg)
// メッセージ ハンドラがありません。
//}}AFX_MSG_MAP
```

```
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CPcpgDlg ダイアログ

CPcpgDlg::CPcpgDlg(CWnd* pParent /*=NULL*/)
	: CDialog(CPcpgDlg::IDD, pParent)
	{
	//{{AFX_DATA_INIT(CPcpgDlg)
		m_distance = 0;
		m_feedrate = 0;
		m_massage = _T("");
	//}}AFX_DATA_INIT
// メモ: LoadIcon は Win32 の DestroyIcon のサブシーケンスを要求
	しません。
	m_hIcon = AfxGetApp()->LoadIcon(IDR_MAINFRAME);
	}

void CPcpgDlg::DoDataExchange(CDataExchange* pDX)
	{
	CDialog::DoDataExchange(pDX);
	//{{AFX_DATA_MAP(CPcpgDlg)
	DDX_Control(pDX, IDC_Xjiku, m_chkX);
	DDX_Control(pDX, IDC_Zjiku, m_chkZ);
	DDX_Control(pDX, IDC_Yjiku, m_chkY);
	DDX_Text(pDX, IDC_EDIT_DISTANCE, m_distance);
	DDX_Text(pDX, IDC_EDIT_FEED, m_feedrate);
	DDX_Text(pDX, IDC_EDIT_MESSAGE, m_massage);
	//}}AFX_DATA_MAP
	}

BEGIN_MESSAGE_MAP(CPcpgDlg, CDialog)
	//{{AFX_MSG_MAP(CPcpgDlg)
	ON_WM_SYSCOMMAND()
	ON_WM_PAINT()
	ON_WM_QUERYDRAGICON()
```

```
ON_BN_CLICKED(IDC_PCPGINIT, OnPcpginit)
ON_BN_CLICKED(IDC_MINUS, OnMinus)
ON_BN_CLICKED(IDC_PLUS, OnPlus)
ON_BN_CLICKED(IDC_PCPGCLOSE, OnPcpgclose)
ON_EN_CHANGE(IDC_EDIT_DISTANCE, OnChangeEditDistance)
ON_EN_CHANGE(IDC_EDIT_FEED, OnChangeEditFeed)
ON_BN_CLICKED(IDC_Xjiku, OnXjiku)
ON_BN_CLICKED(IDC_Yjiku, OnYjiku)
ON_BN_CLICKED(IDC_Zjiku, OnZjiku)
ON_EN_CHANGE(IDC_EDIT_MESSAGE, OnChangeEditMessage)
//}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CPcpgDlg メッセージ ハンドラ

BOOL CPcpgDlg::OnInitDialog()
{
    CDialog::OnInitDialog();

// "バージョン情報..." メニュー項目をシステム メニューへ追加します。

// IDM_ABOUTBOX はコマンド メニューの範囲でなければなりません。
ASSERT((IDM_ABOUTBOX & 0xFFF0) == IDM_ABOUTBOX);
ASSERT(IDM_ABOUTBOX < 0xF000);

CMenu* pSysMenu = GetSystemMenu(FALSE);
if (pSysMenu != NULL)
{
    CString strAboutMenu;
strAboutMenu.LoadString(IDS_ABOUTBOX);
if (!strAboutMenu.IsEmpty())
{
pSysMenu->AppendMenu(MF_SEPARATOR);
pSysMenu->AppendMenu(MF_STRING,
```

```
        IDM_ABOUTBOX, strAboutMenu);
                }
        }

    // このダイアログ用のアイコンを設定します。フレームワークはアプリケーションのメイン
    // ウィンドウがダイアログでない時は自動的に設定しません。
    SetIcon(m_hIcon, TRUE);            // 大きいアイコンを設定
    SetIcon(m_hIcon, FALSE);           // 小さいアイコンを設定

    // TODO: 特別な初期化を行う時はこの場所に追加してください。

    return TRUE;  // TRUE を返すとコントロールに設定したフォーカスは失われません。
    }

void CPcpgDlg::OnSysCommand(UINT nID, LPARAM lParam)
    {
        if ((nID & 0xFFF0) == IDM_ABOUTBOX)
            {
            CAboutDlg dlgAbout;
            dlgAbout.DoModal();
            }
        else
            {
        CDialog::OnSysCommand(nID, lParam);
            }
        }

// もしダイアログボックスに最小化ボタンを追加するならば、アイコンを描画する
// コードを以下に記述する必要があります。MFC アプリケーションは document/view
// モデルを使っているので、この処理はフレームワークにより自動的に処理されます。
```

```
void CPcpgDlg::OnPaint()
{
if (IsIconic())
{
CPaintDC dc(this); // 描画用のデバイス コンテキスト

SendMessage(WM_ICONERASEBKGND, (WPARAM)
dc.GetSafeHdc(), 0);

// クライアントの矩形領域内の中央
int cxIcon = GetSystemMetrics(SM_CXICON);
int cyIcon = GetSystemMetrics(SM_CYICON);
CRect rect;
GetClientRect(&rect);
int x = (rect.Width() - cxIcon + 1) / 2;
int y = (rect.Height() - cyIcon + 1) / 2;

// アイコンを描画します。
dc.DrawIcon(x, y, m_hIcon);
}
else
{
CDialog::OnPaint();
}
}

// システムは、ユーザーが最小化ウィンドウをドラッグしている間、
// カーソルを表示するためにここを呼び出します。
HCURSOR CPcpgDlg::OnQueryDragIcon()
{
return (HCURSOR) m_hIcon;
}

void CPcpgDlg::OnPcpginit()
{
```

```
// TODO:  この位置にコントロール通知ハンドラ用のコードを追加して
                                ください
        //-------------------------------------------------------------
                            // Pcpg46.Dll  を開く
        //-------------------------------------------------------------
            if ( FALSE == pcpg.Pcpg46wDllOpen() ) {
                pcpg.ErrorMessage(PCPG46_BSN_AUTO);
    return ;                        // DLL のロードまたはドライバのオー
                        プンに失敗しました
                                    }
        //-------------------------------------------------------------
                        //  ＤＬＬバージョン情報の表示
        //-------------------------------------------------------------
    if ( FALSE == pcpg.Pcpg46wGetLibVersion(bData)) {
                pcpg.ErrorMessage(PCPG46_BSN_AUTO);
                                    return ;
                                    }
                            wsprintf( szBuf,
                                _T("PCPG46¥n")
                                _T("Dll Version %s"),
                                    bData );
                        pcpg.ResultMessage( szBuf );
        //-------------------------------------------------------------
                        //  ドライババージョン情報の表示
        //-------------------------------------------------------------
    if ( FALSE == pcpg.Pcpg46wGetDrvVersion(bData)) {
                pcpg.ErrorMessage(PCPG46_BSN_AUTO);
                                    return ;
                                    }
                            wsprintf( szBuf,
                                _T("PCPG46¥n")
                                _T("Drv Version %s"),
                                    bData );
                        pcpg.ResultMessage( szBuf );
        //-------------------------------------------------------------
                        //  デバイスの使用を宣言する
```

```
//-------------------------------------------------------------
wBsn = PCPG46_BSN_AUTO;          // 空きデバイス自動検索を指定
        if ( FALSE == pcpg.Pcpg46wCreate(&wBsn) ){
            pcpg.ErrorMessage(PCPG46_BSN_AUTO);
                return ;
                }
            wsprintf( szBuf,
            _T("PCPG46 Created¥n")
            _T("BSN Number %d"), wBsn );
        pcpg.ResultMessage( szBuf );
//-------------------------------------------------------------
            // リソース情報を表示
//-------------------------------------------------------------
    if ( FALSE == pcpg.Pcpg46wGetResource(wBsn,&ri)){
            pcpg.ErrorMessage(wBsn);
                return ;
                }
      wsprintf(szBuf, _T("Board    Name: %s¥n")
                    _T("IO Address：%04Xh-%04Xh¥n"),
                        szPcpg46,

ri.dwIOPortBase[1],ri.dwIOPortBase[1] + ri.dwIOPortLength[1] -
                    1 );
            pcpg.ResultMessage(szBuf);


//-------------------------------------------------------------
            // 第１軸を選択
//-------------------------------------------------------------
            wAxis = PCPG46_AXIS_1;
//-------------------------------------------------------------
            // 制御軸の初期化
//-------------------------------------------------------------
    if ( FALSE == pcpg.Pcpg46_InitAxis(wBsn,wAxis)){
            pcpg.ErrorMessage(wBsn);
                return ;
```

```
                }
            wsprintf( szBuf,
                _T("PCPG46¥n")
        _T("制御軸 %d を初期化しました"), wAxis );
            pcpg.ResultMessage( szBuf );

//-------------------------------------------------------------
                // 第 2 軸を選択
  //-------------------------------------------------------------
            wAxis = PCPG46_AXIS_2;
   //-------------------------------------------------------------
                // 制御軸の初期化
   //-------------------------------------------------------------
if ( FALSE == pcpg.Pcpg46_InitAxis(wBsn,wAxis)){
                pcpg.ErrorMessage(wBsn);
                    return ;
                }
            wsprintf( szBuf,
                _T("PCPG46¥n")
        _T("制御軸 %d を初期化しました"), wAxis );
            pcpg.ResultMessage( szBuf );

//-------------------------------------------------------------
                // 第 3 軸を選択
  //-------------------------------------------------------------
            wAxis = PCPG46_AXIS_3;
   //-------------------------------------------------------------
                // 制御軸の初期化
   //-------------------------------------------------------------
if ( FALSE == pcpg.Pcpg46_InitAxis(wBsn,wAxis)){
                pcpg.ErrorMessage(wBsn);
                    return ;
                }
            wsprintf( szBuf,
                _T("PCPG46¥n")
        _T("制御軸 %d を初期化しました"), wAxis );
```

```
pcpg.ResultMessage( szBuf );

}

void CPcpgDlg::OnMinus()
{
// TODO: この位置にコントロール通知ハンドラ用のコードを追加してください

//-------------------------------------------------------------
  // 一方向に 100*m_distance パルス出力
//-------------------------------------------------------------
  if(m_chkX.GetCheck()){
//-------------------------------------------------------------
  // 第１軸を選択
//-------------------------------------------------------------
  wAxis = PCPG46_AXIS_1;
//-------------------------------------------------------------
  if ( FALSE == pcpg.Pcpg46wDataFullWrite(
wBsn, wAxis, PCPG46_MINUS_PRESET_PULSE_DRIVE,
  1*m_distance ) ) {
  pcpg.ErrorMessage(wBsn);
  return ;
  }
  /*  wsprintf( szBuf,
  _T("PCPG46¥n")
_T("一方向に%d mm 移動しています"), m_distance);
  pcpg.ResultMessage( szBuf );
  */
  }
  if(m_chkY.GetCheck()){
//-------------------------------------------------------------
  // 第２軸を選択
//-------------------------------------------------------------
  wAxis = PCPG46_AXIS_2;
//-------------------------------------------------------------
```

```
        if ( FALSE == pcpg.Pcpg46wDataFullWrite(
wBsn, wAxis, PCPG46_MINUS_PRESET_PULSE_DRIVE,
                1*m_distance ) ) {
                    pcpg.ErrorMessage(wBsn);
                            return ;
                             }
        /*              wsprintf( szBuf,
                          _T("PCPG46¥n")
    _T("一方向に%d mm  移動しています"), m_distance);
                pcpg.ResultMessage( szBuf );
                             */
                            }
                 if(m_chkZ.GetCheck()){
  //-------------------------------------------------------------
                        // 第 3 軸を選択
   //-------------------------------------------------------------
                wAxis = PCPG46_AXIS_3;
   //-------------------------------------------------------------
        if ( FALSE == pcpg.Pcpg46wDataFullWrite(
wBsn, wAxis, PCPG46_MINUS_PRESET_PULSE_DRIVE,
                1*m_distance ) ) {
                    pcpg.ErrorMessage(wBsn);
                            return ;
                             }
        /*              wsprintf( szBuf,
                          _T("PCPG46¥n")
    _T("一方向に%d mm  移動しています"), m_distance);
                pcpg.ResultMessage( szBuf );
                             */
                            }
                    /*      {
                        wsprintf( szBuf,
                          _T("PCPG46¥n")
                    _T("制御軸選択されません"));
                pcpg.ResultMessage( szBuf );
                            return;
```

```
}          */
}

void CPcpgDlg::OnPlus()
{
// TODO: この位置にコントロール通知ハンドラ用のコードを追加してください

    //-------------------------------------------------------------
    // ＋方向に m_distance パルス出力
    //-------------------------------------------------------------

    if(m_chkX.GetCheck()){
    //-------------------------------------------------------------
    // 第１軸を選択
    //-------------------------------------------------------------
    wAxis = PCPG46_AXIS_1;
    //-------------------------------------------------------------
    if ( FALSE == pcpg.Pcpg46wDataFullWrite(
    wBsn, wAxis, PCPG46_PLUS_PRESET_PULSE_DRIVE,
    m_distance ) ) {
    pcpg.ErrorMessage(wBsn);
    return ;
    }
    /*  wsprintf( szBuf,
    _T("PCPG46¥n")
    _T("＋方向に%d mm 移動しています"),m_distance);
    pcpg.ResultMessage( szBuf );
    */
    }
    if(m_chkY.GetCheck()){
    //-------------------------------------------------------------
    // 第２軸を選択
    //-------------------------------------------------------------
    wAxis = PCPG46_AXIS_2;
    //-------------------------------------------------------------
```

```
        if ( FALSE == pcpg.Pcpg46wDataFullWrite(
    wBsn, wAxis, PCPG46_PLUS_PRESET_PULSE_DRIVE,
                    m_distance ) ) {
                      pcpg.ErrorMessage(wBsn);
                                  return ;
                                   }
            /*              wsprintf( szBuf,
                              _T("PCPG46¥n")
      _T("＋方向に%d mm  移動しています"),m_distance);
                  pcpg.ResultMessage( szBuf ); */
                                  }
                    if(m_chkZ.GetCheck()){
     //-------------------------------------------------------------
                          // 第 3 軸を選択
       //-------------------------------------------------------------
                    wAxis = PCPG46_AXIS_3;
       //-------------------------------------------------------------
        if ( FALSE == pcpg.Pcpg46wDataFullWrite(
    wBsn, wAxis, PCPG46_PLUS_PRESET_PULSE_DRIVE,
                    m_distance ) ) {
                      pcpg.ErrorMessage(wBsn);
                                  return ;
                                   }
            /*              wsprintf( szBuf,
                              _T("PCPG46¥n")
     _T("＋方向に%d  パルス  移動しています"),m_distance);
                  pcpg.ResultMessage( szBuf );
                                   */
                                  }
                          /*      {
                      wsprintf( szBuf,
                              _T("PCPG46¥n")
                    _T("制御軸選択されません"));
                pcpg.ResultMessage( szBuf );
                                  return;
                                  }
```

```
*/
}

void CPcpgDlg::OnPcpgclose()
{
// TODO: この位置にコントロール通知ハンドラ用のコードを追加してください

    //------------------------------------------------------------
    // デバイスを解放する
    //------------------------------------------------------------
    if ( FALSE == pcpg.Pcpg46wClose(wBsn)){
        pcpg.ErrorMessage(wBsn);
        return ;
    }
    wsprintf( szBuf,
    _T("PCPG46 Close¥n")
    _T("Bsn %d"), wBsn );
    pcpg.ResultMessage( szBuf );
    //------------------------------------------------------------
    // Pcpg46.Dll を閉じる
    //------------------------------------------------------------
    if ( FALSE == pcpg.Pcpg46wDllClose()) {
        pcpg.ErrorMessage(PCPG46_BSN_AUTO);
return ;                                                    // DLL のアンロードに失敗しました
    }
    return ;

}

void CPcpgDlg::OnChangeEditDistance()
{
// TODO: これが RICHEDIT コントロールの場合、コントロールは、lParam マスク
// 内での論理和の ENM_CHANGE フラグ付きで
```

```
CRichEditCrtl().SetEventMask()
// メッセージをコントロールへ送るために CDialog::OnInitDialog()
関数をオーバー
// ライドしない限りこの通知を送りません。

// TODO: この位置にコントロール通知ハンドラ用のコードを追加して
ください
CEdit * str = (CEdit *) GetDlgItem(IDC_EDIT_DISTANCE);
char count[10];
int n;
n=str->GetLine(0,count,10);
count[n] = 0;
m_distance = atoi(count);
}



void CPcpgDlg::OnChangeEditFeed()
{
// TODO: これが RICHEDIT コントロールの場合、コントロールは、
lParam マスク
// 内での論理和の ENM_CHANGE フラグ付きで
CRichEditCrtl().SetEventMask()
// メッセージをコントロールへ送るために CDialog::OnInitDialog()
関数をオーバー
// ライドしない限りこの通知を送りません。

// TODO: この位置にコントロール通知ハンドラ用のコードを追加して
ください
CEdit * str = (CEdit *) GetDlgItem(IDC_EDIT_FEED);
char count[10];
int n;
n=str->GetLine(0,count,10);
count[n] = 0;
m_feedrate = atoi(count);
if(m_chkX.GetCheck()){
```

```
//-------------------------------------------------------------
// 第１軸を選択
//-------------------------------------------------------------
wAxis = PCPG46_AXIS_1;
//-------------------------------------------------------------
if ( FALSE ==
pcpg.Pcpg46_Change_Feed(wBsn,wAxis,m_feedrate)){
pcpg.ErrorMessage(wBsn);
return ;
}
wsprintf( szBuf,
_T("PCPG46¥n")
_T("制御軸 %d のパルス発生速度を%d PPS に変換しました"),
wAxis,m_feedrate);
pcpg.ResultMessage( szBuf );
}else
if(m_chkY.GetCheck()){
//-------------------------------------------------------------
// 第 2 軸を選択
//-------------------------------------------------------------
wAxis = PCPG46_AXIS_2;
//-------------------------------------------------------------
if ( FALSE ==
pcpg.Pcpg46_Change_Feed(wBsn,wAxis,m_feedrate)){
pcpg.ErrorMessage(wBsn);
return ;
}
wsprintf( szBuf,
_T("PCPG46¥n")
_T("制御軸 %d の送り速度を%d mm/min に変換しました"),
wAxis,m_feedrate);
pcpg.ResultMessage( szBuf );
}else
if(m_chkY.GetCheck()){
//-------------------------------------------------------------
// 第 3 軸を選択
```

```
//-------------------------------------------------------------
                wAxis = PCPG46_AXIS_3;
//-------------------------------------------------------------
                    if ( FALSE ==
pcpg.Pcpg46_Change_Feed(wBsn,wAxis,m_feedrate)){
                        pcpg.ErrorMessage(wBsn);
                            return ;
                              }
                        wsprintf( szBuf,
                          _T("PCPG46¥n")
        _T("制御軸  %d  の送り速度を%d mm/min に変換しました"),
                      wAxis,m_feedrate);
                    pcpg.ResultMessage( szBuf );
                            }else
                              {
                        wsprintf( szBuf,
                          _T("PCPG46¥n")
                      _T("制御軸選択されません"));
                    pcpg.ResultMessage( szBuf );
                            return;
                              }

                            }



                    void CPcpgDlg::OnXjiku()
                              {
// TODO:  この位置にコントロール通知ハンドラ用のコードを追加して
                              ください
                            A=1;
                              }

                    void CPcpgDlg::OnYjiku()
                              {
// TODO:  この位置にコントロール通知ハンドラ用のコードを追加して
```

```
ください
A=2;
    }

void CPcpgDlg::OnZjiku()
{
// TODO: この位置にコントロール通知ハンドラ用のコードを追加して
ください
A=3;
    }

void CPcpgDlg::OnChangeEditMessage()
{
// TODO: これが RICHEDIT コントロールの場合、コントロールは、
lParam マスク
// 内での論理和の ENM_CHANGE フラグ付きで
CRichEditCrtl().SetEventMask()
// メッセージをコントロールへ送るために CDialog::OnInitDialog()
関数をオーバー
// ライドしない限りこの通知を送りません。

// TODO: この位置にコントロール通知ハンドラ用のコードを追加して
ください

}
```