計測自動制御学会東北支部第 202 回研究集会(2002.07.02)
資料番号 202-8

データフロー処理方式を利用した機能分散制御システム

The Functional Distributed Control System Using a Dataflow Processing Scheme

○趙　丹寧*　　吉岡　良雄**
○Dan-Ning ZHAO* , Yoshio YOSHIOKA**
*弘前大学・大学院理学研究科・情報科学専攻
**弘前大学・理工学部・電子情報システム工学科
*Graduate School of Science,Hirosaki University
**Department of Electrical and Information System Engineering,
Faculty of Science and Technology,Hirosaki University

キーワード： 機能分散制御(functional distributed control),
単方向ループ状接続方式(uni-directional loop connection),
データフロー処理方式(dataflow processing scheme),

連絡先：〒360-8561　弘前市文京町 3 弘前大学理工学部 電子情報システム工学科,
吉岡　良雄,Tel(0172)39-6133,E-mail:{gs01615,siyoshi}@si.hirosaki-u.ac.jp

1. はじめに

ベルトコンベアでの製造用ロボットのような大規模制御システムでは、多くのセンサ部からの入力データを処理し、多くの制御部へ制御データを出力する必要がある。これらの入力処理、データ解析処理および出力制御を一つのコンピュータで行うことは、そのソフトウエアが非常に複雑なものになる。さらに、センサ部や制御部の変更のたびに、ソフトウエアの変更が必要になる。そこで、現在では、ソフトウエアの変更を少なくするために、入出力部とのインタフェースを規定し、各センサ部および各制御部に小さなプロセッサを置いて、そのインタフェース処理を行う制御システムを構築する方向である。しかしながら、この場合の処理でも、それぞれのセンサ用プロセッサからのデータを中央のコンピュータ（ホストコンピュータという）に送って、そのホストコンピュータでデータ解析処理を行い、その結果に応じて、それぞれの制御プロセッサへ制御データを送る方法である。システムが大規模になればセンサ部や制御部が多くなり、ホストコンピュータの負荷も多くなる。そして、そのソフトウエアもさらに複雑なものになる。

一方において、マイクロプロセッサの機能が向上し、これを利用したロボット（制御システム）が数多く発表されている 1)。このようなロボットにおいても、多くのセンサからの入力信号や多くの制御信号がある。多くの入出力信号を一つのマイクロプロセッサで処理する場合、回路およびプログラムが非常に複雑なものになってしまう。そこで、一つの機能（センサからの入力処理や制御処理など）しか持たない小さなマイクロプロセッサと、メインの処理プロセッサ（ホス

トコンピュータという）を共通バスで接続した機能分散処理システムがある 2)。しかしながら、ファンナウトにより、12 プロセッサまでしか接続できず、大規模システムには適用できない。また、パケット衝突の回避を行うプログラムが複雑なものになっている。

以上のことから、複数のプロセッサを利用して機能分散制御システムを構築する場合、次の条件が必要であろう。すなわち、

(1) 複数の入力プロセッサおよび出力プロセッサのネットワーク接続が単純であり、かつ通信方式(プロトコル)も単純であること

(2) プロセッサの増設が容易であり、増設による各プロセッサのプログラム変更も容易であること,

(3) 故障プロセッサを容易に削除でき、故障プロセッサの影響が他に波及しないこと

(4) システム全体を制御するプログラムは、並列処理や通信処理を意識することなく作成できることである。

これらの条件のうち、条件 (1) のネットワーク接続としては、入力と出力一対のポートを利用する方法であり、共通バス方式および単方向性ループ状接続方式があげられる。前者は、100 プロセッサ以上接続を考えた場合、ファンナウトやデータブロック（パケット）の衝突などの問題がある。これらに対処するハードウエア（バスアービタなどの回路）は複雑なものになってしまう。これに対して、後者の単方向性ループ状接続では、これらの問題が起こらない。条件(2)は、共通バス方式および単方向性ループ状接続方式の双方とも満足する条件である。また、条件(3)は、単方向性ループ状接続方式において工夫が必要である。条件(4)は、複数のプロセッサによる並列処理や通信処理を意識しない新たな処理方式を考える必要がある。この処理方式の例として、データフロー処理方式があげられる。

そこで、ワンチップマイクロプロセッサに内蔵されている直列通信ポートを利用して単方向性ループ状接続を行い、内蔵されている並列ポートでセンサからの入力や制御出力を行う機能分散制御システムを提案した。機能分散制御システムは、ハード的に単純で済むので、現存するワンチップマイクロプロセッサを用いて容易に構築できる。従って、安価なシステム構築が可能である。また、本システムでは、ホストコンピュータを介さずにセンサ部および制御部のプロセッサで簡単な処理を行うことを目的としている。この処理方法として、データフロー処理方式を採用する。これによって、上述したようにプロセッサ間の複雑な通信を意識することなく、プログラミングが可能である。従って、従来のようにアセンブリ言語やC言語などを利用して各プロセッサのプログラミングを行う場合に比べ、プログラミングが容易である。

本論文は、まず機能分散制御システムの一般的構成、データフロー処理方式3)を実装したプロトタイプ機のアーキテクチャ仕様、マイクロプロセッサの故障等に対する切り離し手法などを示す。さらに、簡単なデータフロー言語による制御プログラムを作成するとともに、機能分散制御システムのプロトタイプ機の構成と仕様を示した上で、プロトタイプ機について入出力信号に対する応答時間を測定するシステムの構成とその実験結果について考察する。

2. アーキテクチャ仕様

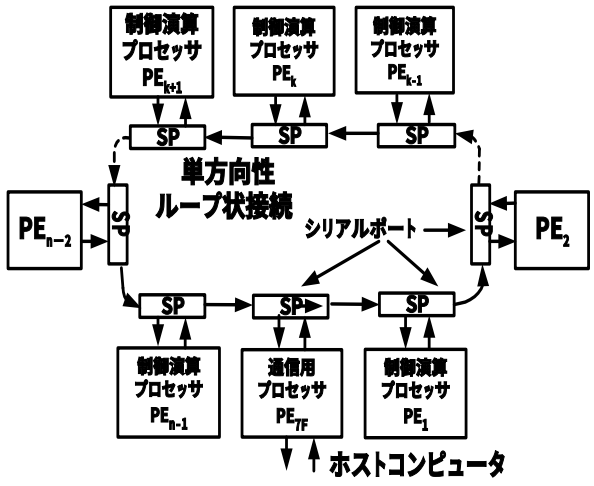


図 1　機能分散制御システムの構成

2.1　システム全体の構成

機能分散制御システムは、図１に示すように、複数のプロセッサを直列転送ポート(SP:Serial Port)によって単方向性ループ状接続を行ったシステムである。各処理・制御プロセッサ(PE_k)は、簡単な処理を行うとともに、並列ポートを利用してセンサからの入力または外界に対する制御を行う。すなわち、各処理・制御プロセッサは同じハードウエア構成であり、入力・出力一対の直列通信用ポートおよび入力・出力一対の並列ポートをもっている。また、通信用プロセッサ(PE_{7F})はホストコンピュータと接続されている。プロトタイプ機では、二つの直列通信用ポートおよび一つの入出力並列ポートなどを内蔵した Z80 のワンチップLSI（Z84C015）を利用している。

次に、処理プログラムについて、各処理・制御プロセッサが利用しているアセンブリ言語（機械語）を用いてプログラミングを行う場合は、各処理・制御プロセッサ個々にプログラミングを行う必要がある。従って、プロセッサ間のデータ交換を行うプログラミングが非常に煩雑になってしまう。そこで、各処理・制御プロセッサの制御プログラム(OS)が通信プログラムを含めてまったく同じであれば、そのプログラミングは容易となる。このような考えのもとで、プログラミングレベルで複数のプロセッサ間での通信を意識することがないデータフロー処理方式を採用する。

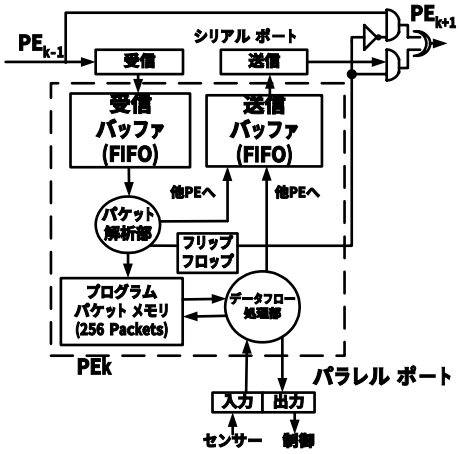


図 2　処理・制御プロセッサの内部構成

2.2　処理・制御プロセッサの構成

プロトタイプ機における各処理・制御プロセッサの内部構成は、図２に示すようになっている。図において、直列通信ポート(Serial Port)の出力側回路は、故障プロセッサを切り離す回路である（後述）。ここで、プロセッサ間の通信は、図３に示すような８バイト固定長パケット（転送パケット）によって行う。また、データフロー処理方式を実現するプログラムパケットは図４に示すように32バイトからなっており、各処理・制御プロセッサに256個の領域をもっている。この領域には、上述の８バイト固定長パケットによって格納される。なお、転送パケットおよびプログラムパケットの各領域の意味は次のようになる。まず、DPE（Destination PE）および SPE（Source PE）はそれぞれ目的プロセッサ番号（0x01-0x7F）および源プロセッサ番号（0x01-0x7F）である。特に、0x7F の場合、通信用プロセッサ（または、ホストコンピュータ）を表す。また、定数データパケットおよび変数データパケットにおいて、DPE の最上位ビットが０のときデータ領域A を、１のときデータ領域B を指定し、データ

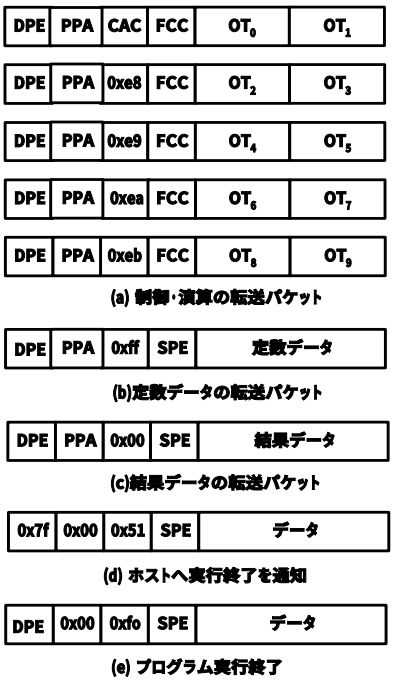


図 3　転送パケット形式例（8 バイト固定長）

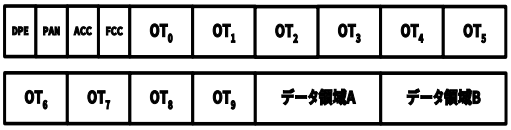


図 4　データフロー処理用プログラムパケット例

(32 バイト固定長)

をそれぞれのプログラムパケット領域に格納する。次に、PPA (Program Packet Area Number) (0～255) は、各処理・制御プロセッサにおけるプログラムパケットの格納領域番号である。また、CAC (Control and Arithmetic Code) はポートからの入出力制御、4 バイト整数や実数の演算などのコードである。そして、FCC (Fire Condetion Code) は発火条件コードであり、そのビット $(00AB00ab)_2$ において、ab は発火条件フラグを、AB はデータ到着フラグを表す。OT_n 領域は変数 (結果) データパケットを生成するときの目的プロセッサ番号 DPE、とプログラムパケットの領域番号 PPA を示す。ここで、目的プロセッサ番号において、最上位ビットが 0 のときデータ領域 A を、1 のときデータ領域 B を指定する。さらに、データ領域 A およびデータ領域 B は演算データが格納される領域であり、整数または実数 4 バイトが格納される。

次に、転送パケットの受信処理は次のようにして行われる。図 2 において、直列通信ポートから割り込みによって 1 バイトずつ受信し、受信バッファに格納する。そして、パケット解析部において、受信バッファから 1 バイトずつ取り込み、パケットを生成する。そのパケットの DPE 領域を解析して、そのパケットが他プロセッサ宛てであれば、送信バッファに格納する。そのパケットが自プロセッサ宛てであれば、PPA が指定するプログラムパケットの該当領域に格納する。受信パケットが定数データパケットまたは変数 (結果) データパケットである場合、データを指定するデータ領域に格納するとともに、発火条件コード FCC の発火条件フラグ ab をリセット、またはデータ到着フラグ AB をセットする。

また、直列通信ポートからのパケット送信は、タイマー割り込みによって、送信バッファに送信パケットがあれば、送信バッファから 1 バイトずつ取り出して、次のプロセッサへ送信する。

2.3　データフロー処理方式の実現

データフロー処理方式は以下のようにして進められる。まず、ホストコンピュータから、図 3 (a) に示す転送パケットによって、各処理・制御プロセッサのプログラムパケットメモリにプログラムパケットが形成される。データフロー処理部は、このプログラムパケットの発火条件コード FCC をチャックし、発火条件フラグ ab とデータ到着フラグ AB が一致したとき (これを発火という)、そのプログラムパケットの制御・演算コード CAC に従って、演算データ A および B について処理を行う。そして、最大 10 個の出力先 (OT_n) への変数 (結果) データパケットを生成する。生成された変数 (結果) データパケットが他プロセッサへのパケットであれば送信バッファに書き込み、自プロセッサ宛であれば該当プログラムパケット領域に結果データを書き込むとともに FCC のデータ到着フラグ AB をセットする。なお、他プロセッサから受け取った自プロセッサ宛の変数データパケットも、同様にパケット解析部において該当プログラムパケット領域に結果データを書き込むとともに FCC のデータ到着フラグ AB をセットする。

2.4　故障プロセッサの切り離し

図 2 に示す故障プロセッサ切り離し回路は、以下のように動作する。まず、システムがリセットされた (または、電源が入った) とき、フリップフロップ FF から出力される信号は Low 状態である。このとき、前プロセッサの直列通信ポートからの出力信号は出力側に出力され、次のプロセッサに送られる。すなわち、このプロセッサの直列通信ポートからの出力信号は、次にプロセッサへ出力されないことになる。この状態において、ホストコンピュータからこのプロセッサへの故障診断パケットが送られると、直列通信ポートの

受信部からはこのパケットを受信することができ、受信バッファに入力される。そこで、このプロセッサが正常に動作する場合は、パケット解析部からフリップフロップFFをセット状態にして、フリップフロップFFの出力信号をHighにする。このとき、このプロセッサの直列通信ポートからの出力信号が出力側に出力できるようになり、これとともに、故障診断パケットに対する応答パケットをホストコンピュータへ返送する。もし、このプロセッサに故障がある場合には、フリップフロップFFからの出力信号をLowのままとし、故障診断パケットに対する応答パケットを出力しない。従って、ホストコンピュータは、プロセッサからの応答パケットの“あり・なし”によって、故障プロセッサを検出することができる。ただし、ホストコンピュータから各プロセッサへ送られた故障診断パケットはホストコンピュータに戻ってくる。しかし、応答パケットではないので、ホストコンピュータはこれを無視する。

```
NO:  source program
 1:  ;  test program
 2:   pe(1)
 3:  fx=in(fs)
 4:   if(fx) fdd,fdd,fss
 5:  fxx=set(fx,fss)
 6:  fa=set(3,fss)
 7:  ft5=0.001-ft4
 8:   if(ft5) fd1,fd2,fd2
 9:  fa1=0.5*ft2
10:  ft3=fa1-fa
11:  ft4=abs(ft3)
12:   return(fdd)
13:   pe(3)
14:  fxx=set(fxx,fd1)
15:  fa=set(fa1,fd1)
16:  fans=set(fa1,fd2)
17:  fs=out(fans)
18:  ft1=fxx/fa
19:  ft2=fa+ft1
20:  fs=start
21:   end
```

図5　プログラム例

2.5　プログラミング言語

データフロー処理をパケットレベルでプログラミングを行うことは、非常に困難である。そこで、文字列を利用して、データフロー処理のプログラミングを行うことができるプログラミング言語（簡易言語）を開発した。この言語を用いて2乗根を計算するプログラムの例は,図5のように記述することができる。図の2行目および13行目のpe(n)は、以降に記述されるプログラムを格納する演算・制御プロセッサ番号nを指定する文である。また、X=in(A),X=set(A,B),X=abs(A),return(A),X=out(A)は、それぞれホストコンピュータからデータを入力しXへ出力する文、データAおよびデータBが揃ったらAをXに出力する文、Aの絶対値をXに出力する文、データAが到着したときプログラムを終了する文、ホストコンピュータへデータAを出力するとともにAをXに出力する文である。この他に、指数計算や対数計算を行う文がある。4行目および8行目のif(A) DM,DZ,DP文は、データAをテストし、負であればDMへ、ゼロであればDZへ、正であればDPへデータAを出力する文である。8行目および10行目のX=A-Bや9行目のX=A*Bなどは4バイト整数演算または4バイト実数演算の文である。さらに、A=startは初期データ0をAに強制的に出力する、プログラム実行開始文である。このプログラムの例では、分かりやすいように上から下へデータが流れるように記述してあるが、データフロー処理の性格上、各文の記述順序は自由である。これらの文は、図4に示すプログラムコードに変換され,図3の転送パケットによって各演算・制御プロセッサのプログラムパケット領域に格納される。

2.6　入力処理・制御処理

前節のプログラミング言語において、並列ポートからのデータ入力および並列ポートへのデータ出力は、図6に示すように、それぞれx1=inp(a)およびx2=outp(a1)文によって記述される。そして、データ入力は整数x1の下位8ビットに格納され、データ出

力は整数a1の下位8ビットが並列ポートへ出力される。図において、データaが入力されると並列ポートからデータが入力されx1の下位8ビットに入力される。また、データbが入力されると、並列ポートへ$(05)_{16}$が出力される。さらに、並列ポートにセンサなどからの外部データが入力されると、そのデータが割り込みルーティンによってcの下位8ビットに格納され、データdが到着するとcの値がx3に出力される。ここで、dを定数データにしておけば、センサからのデータ入力の毎に、cの値が入力され、即処理されることになる。

```
NO:  source program
  1:  ;  input
  2:  x1=inp(x)
  3:  -----------
  4:  ;  output
  5:  a1=set($05,b)
  6:  x2=outp(a1)
  7:  -----------
  8:  ;  interrupt input
  9:  x3=inpi(c,d)
 10:  -----------
```

図6　並列ポートへの入力・出力文

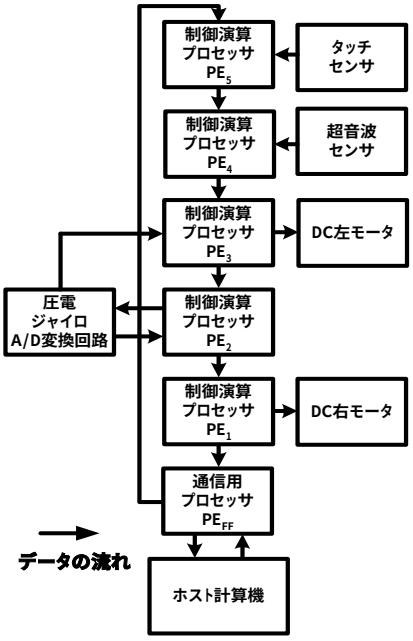


図7　プロセッサと制御装置の対応図

3.　プロトタイプ機

機能分散制御システムのプロトタイプ機(機能分散制御ロボット)を構築した。このプロトタイプ機はロボットの制御を目的として構成しており、制御および入力としてDCモータやセンサなどがある。そのプロセッサと制御装置の対応は図7に示すようになっている。ここでは、一応各処理・制御するプロセッサはこのように割り当ててあるが、実際に各処理・制御プロセッサは同じハードウェア構成であるため、どのプロセッサでも制御することが可能である。

このロボットは、図8に示すように、電源に12V仕様のバッテリーを用い、駆動方式はギヤボックスを組み合わせた12V仕様のDCモータを用いた。センサ類にはタッチセンサ、超音波センサなどを使用した。ロボットの仕様を以下に示す。

図8　ロボット(プロトタイプ機能分散制御システム)

✧　マイクロプロセッサ部

CPU :Z80　4.9152MHz

RAM :32KB

ROM :32KB

シリアルポート：　A,Bチャンネル

通信速度　：RS-232C　9600bps

Aチャンネル：ホスト計算機への通信用

(通信用プロセッサのみ)

Bチャンネル：各プロセッサへの通信用
パラレルポート:　A,Bチャンネル
Aチャンネル：制御対象からの入力
Bチャンネル：制御対象からの出力
タイマー：割り込み間隔1.25ms

✧　ホスト計算機
CPU :PentiumIII400MHz,RAM :192MB
周辺機器 :4入力USBハブ、無線LAN(USB)、
CCDカメラ(USB)

✧　センサ部
タッチセンサ
超音波センサ

✧　姿勢制御部
圧電ジャイロ

✧　電源部
自動二輪用12Vバッテリー
自動車用DC-ACコンバータ　300W規格

4　評価実験

データフロー処理方式を利用した機能分散制御システムを評価するために、今回は機能分散制御システムのプロトタイプ機における、入出力信号に対する応答時間を測定する実験を行った。計測システムの構成とその評価結果について以下に述べる。

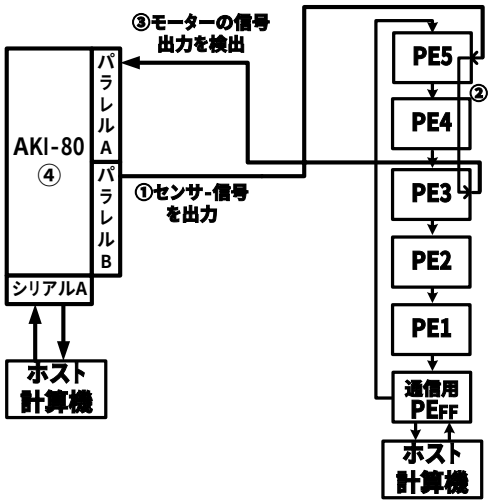


図9　計測システムの構成図

4.1　目的

ロボットにおいてセンサ等の入力から事象を判断し、その事象に対する出力までの時間はロボットシステムとしては重要な要素の一つである。例えば、ロボットが走行中、障害物に衝突する場合を考えたとき、可能ならば衝突前に障害物を察知し衝突を防ぎたいものである。そのため、今回は衝突があった場合を想定し、タッチセンサ入力からDCモータ停止までの応答時間を測定し、評価を行う。ここで、本システムによる構成の特徴から、通信するプロセッサ数が応答時間に比例することが予想される。そこで、今回の実験では単純にタッチセンサの代わりとなる入力信号をタッチセンサ担当のプロセッサに入れ、適当なプロセッサを経由して、DCモータ担当のプロセッサからの出力信号が検出されるまでの時間を計測し、これをタッチセンサ入力からDCモータ停止までの応答時間として測定を行う。

4.2　計測システムの構成

図9に示すように、プロトタイプ機は6プロセッサからなっている。そして、タッチセンサからPE5へ入力する信号の変わりに測定システムのマイコン(AKI-80)のパラレルBポートから信号を出力する。出力と同時にAKI-80タイマーカウンタを動作させ、カウント開始する。次に、AKI-80からの信号を出力すると、PE5はセンサの信号が入力されると認識し、PE3へDCモータ停止のパケットを送信する。そして、PE3にパケットが到着するとDCモータ停止信号として信号を出力する。この停止信号がAKI-80のパラレルAポートに入力されると同時にタイマーカウントを停止する。そのカウント値をAKI-80に接続されているパソコンへ出力する。最後に、以上の流れを9990回実行する。なお、信号入力判定などの処理は他のプロセッサで行う。

このプロトタイプ機のパケット送信はタイマー割り込みによって1byteずつ送信できるように、RS-232C 9600bpsで、1.25ms毎のタイマー割り込みになっている。また、データ1byteにつき3ビット（ス

タートビット 1bit、ストップビット 2bit）付加されるので、1 バイトの送信において 1bit 分の余裕がある。この間隔で 1 パケット（同期バイトを含め 10 バイト）を送信すると、1 プロセッサ間の通信に必要な時間は最小1.25*ms*×10(パケット)＝12.5*ms* である。

また、測定システムのタイマー割り込み周期は 0.25ms に設定してある。

4.3 応答時間測定実験

(1) 実験1

PE5 から入力、PE4 で入出力有無を判断、PE3 で出力する 2 プロセッサをパケットが転送される場合である（図11)。その場合のデータフロープログラムは図10 のようになっている。その測定結果は図13 のようになる。その応答時間の平均値は83.6ms である。

```
;センサ判定、前進、停止
;センサ判定
 pe(5)
x00=inp(s)
 pe(4)
 out(nto)
x01=x00-62
 if(x01)nt4,nt4,nt0
 pe(3)
;前進
m1=set(1,nt4)
s=outp(m1)
;停止
m2=set(0,nt0)
s=outp(m2)
s=start
 end
```

図10　データフロープログラム

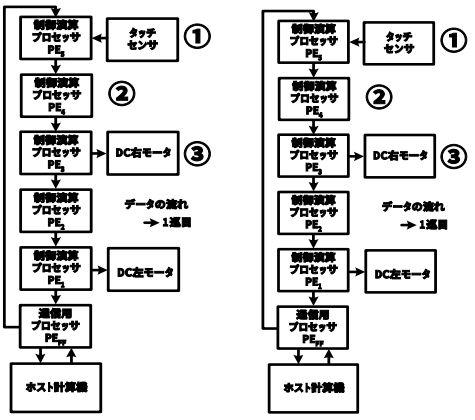


図 11　実験1 のプロセッサ構成　　図 12　実験2 のプロセッサ構成

(2) 実験2

PE5 から入力、PE4 で入出力の有無を判断、PE1 で出力する 4 プロセッサをパケットが転送される場合である（図12)。その測定結果は図14 のようになる。その応答時間の平均値は83.7ms である。

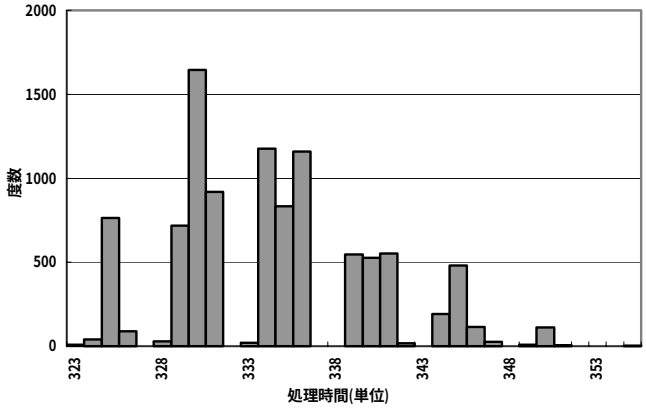


図13　実験1（2プロセッサ経由）

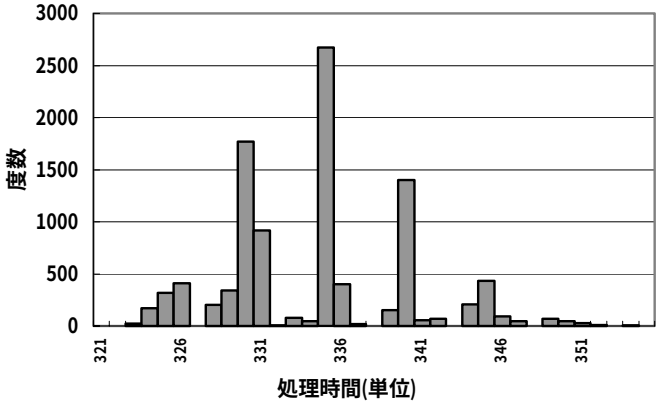
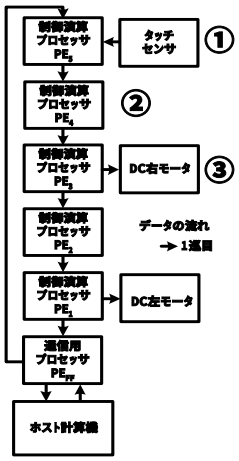


図14　実験2（4プロセッサ経由）

(3) 実験3

PE4 から入力、PE5 で入出力の有無を判断、PE3 で出力する 7 プロセッサをパケットが転送される場合である（図15)。その測定結果は図17 のようになる。その応答時間の平均値は163.3ms である。



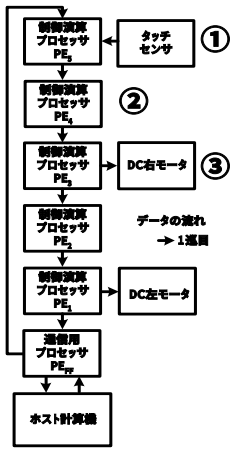


図 15　実験3のプロセッサ構成　　図 16　実験4のプロセッサ構成

(4)　実験4

PE5 から入力、PE1 で入出力の有無を判断、PE3 で出力する 8 プロセッサをパケットが転送される場合である（図16）。その測定結果は図18のようになる。その応答時間の平均値は163.2msである。

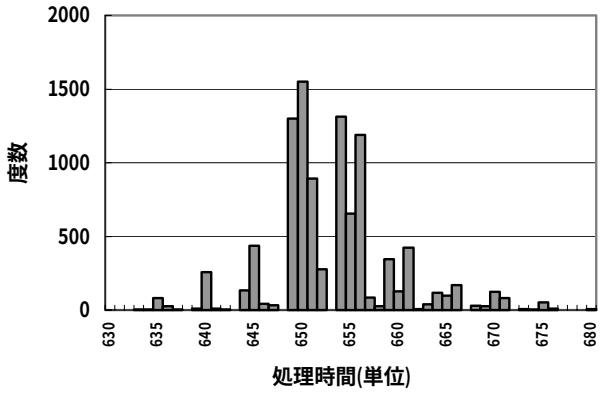


図17　実験3（7プロセッサ経由）

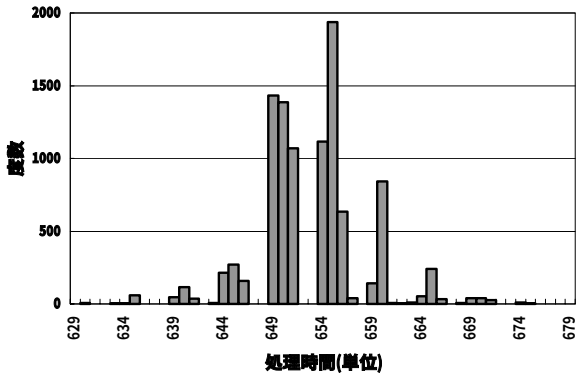


図18　実験4（8プロセッサ経由）

(5)　実験5

PE4 から入力、PE5 で入出力の有無を判断、PE1 で出力する 9 プロセッサをパケットが転送される場合である（図19）。その測定結果は図20のようになる。その応答時間の平均値は163.3msである。

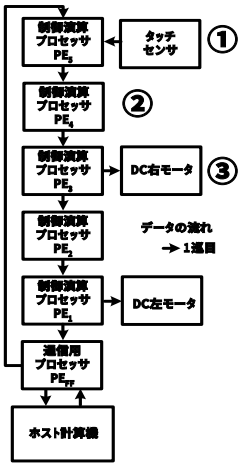


図19　実験5のプロセッサ構成

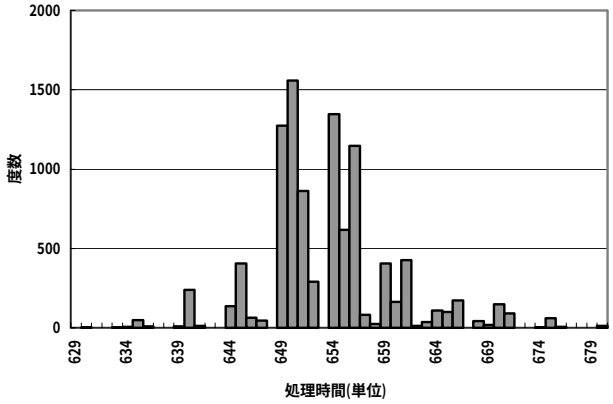


図20　実験5（9プロセッサ経由）

4.4　実験結果の考察

これらの実験結果について検討を行うと、実験1と2の応答時間は、ほぼ同じ値であり、実験3～5においてもほぼ同じ値になっている。そこで、各プロセッサのパケット受信信号および測定システムの入出力信号をロジックアナライザで測定を行い、その結果は図 21 のようになった。この図から、パケットが上から下へ順次転送されていくことが分かる。そして、最下位の信号がHigh の期間が測定システムの応答時間として測定した

時間である。この結果および AKI-80 の測定プログラムの検討から実験1と2の測定時間は1パケットが6プロセッサを1周する時間になっており、実験 3～5 の測定時間は2周する時間になっていることがわかった。また、1 パケットを 1 プロセッサに送る時間は、12.5ms～13.8ms の範囲であることも分かった。従って、応答時間は、モータ出力プロセッサからセンサプロセッサへの転送パケットの転送時間を差し引いた値で求められる。すなわち、

実験1での平均応答時間は

83.6ms－13.125ms×4=31.1ms、

実験2での平均応答時間は

83.7ms－13.125ms×2=57.45ms、

実験3での平均応答時間は

163.3ms－13.125ms×5=97.675ms、

実験4での平均応答時間は

163.2ms－13.125ms×4=110.7ms、

実験5での平均応答時間は

163.3ms－13.125ms×3=123.925ms

となる。

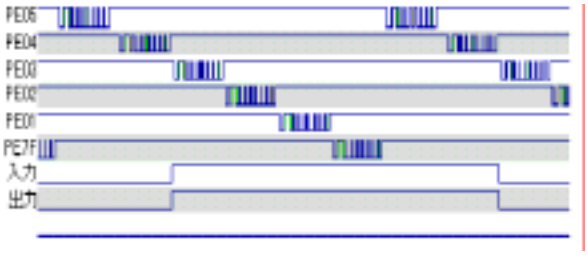


図21　ロジックアナライザ波形

5.　まとめ

大規模制御システムでは、多くのセンサ部からの入力データを処理し、多くの制御部へ制御データを出力する必要がある。これらの入力処理、データ解析処理および出力制御を一つのコンピュータで行うことは、そのソフトウエアが非常に複雑なものになる。そこで、マイクロプロセッサに内蔵されている並列ポートでセンサからの入力や制御出力を行う機能分散制御システムを提案した。また、提案した本システムは、データフロー処理方式を採用して、ホストコンピュータを介さずにセンサ部および制御部のプロセッサで簡単な処理を行うことが可能である。本論文では、機能分散制御システムの一般的構成を示し、簡単なデータフロー言語によるプログラムを作成した。さらに、データフロー処理方式を採用した機能分散制御システムのプロトタイプ機について入出力信号に対する応答時間を測定する実験を行った。今後の課題として、システムの規模を大きくし、構成の違いによる応答時間の変化を考察と検討を行う。更により多くの構成パターンによる実験を行う。

謝辞

本報告を進めるにあたり、ご指導を頂きました成田明子 助教授、一條　健二　助手、そして、OB の佐々木　孝仁　先輩に心から感謝いたします。

参考文献

[1]　たとえば“HONDA ASIMO”, http://www.honda.co.jp/ASIMO/

[2]　“RoboCube”,http://www.watt.co.jp/

[3]　吉岡　良雄:“図解　コンピューターアーキテクチャ入門”pp.99-121,オーム社,(1992)

[4]　佐々木　孝仁,吉岡　良雄　“機能分散制御システムの構築”,平成 13 年度電気関係学会東北支部連合大会