計測自動制御学会東北支部 第192回研究集会 (2000.12.12)
資料番号 192-4

PC/AT互換機を用いた４脚ロボットの制御環境の構築

Development of control system of quardruped robot on PC/AT using Linux and Universal Interface Board

○熊谷正朗*，小関篤史*，江村超*

○Masaaki Kumagai*, Atsushi Koseki*, Takashi Emura*

*東北大学大学院工学研究科

*Tohoku University, Faculty of Engineering

キーワード： ４脚ロボット (quardruped robot)，Linux(linux)，インターフェイス(interface)，CPLD(complex programmable logic device)，PCI拡張ボード (PCI interface card)

連絡先： 〒980–8579　仙台市青葉区荒巻字青葉０１　東北大学大学院工学研究科機械電子工学専攻 江村研究室
熊谷正朗，Tel.: (022)217–6969，Fax.: (022)217–6967　E-mail: kumagai@emura.mech.tohoku.ac.jp

1. はじめに

今日のコンピュータの性能向上は著しく，数年前は数台のコンピュータに分散しなければならなかったような処理が，１台のコンピュータで余裕をもって処理できるようになった．また，パーソナルコンピュータ(PC)の普及率増加に伴い，低価格化も進んでいる．それと同時に，同じ部品を使いながら，基板一枚にPC並の機能を持たせた組込用のコンピュータも多数市販され，移動ロボットの自立化を行うことも非常に容易となった．

しかしながら，PCの急速な技術進歩は，ロボット制御に利用するには不便な面も出てきた．PCの拡張コネクタにおいて，数年前までは，Cバス，ISAバスといった，ロボット制御に必要なインターフェイス回路を容易に自作可能な仕様のバスが主流であったが，近年のPCは，自作では対処困難なPCIバスにほとんど移行してしまった．また，コンピュータそのものの性能向上により，ロボット制御には便利であった MS–DOSでは，その機能を全く生かしきることが出来なくなった．

以上のような背景の下，著者らは今日の高性能PCをロボット制御に利用するために有用な技術開発を行った．あえてPCにこだわるのは，その使い勝手と入手製の良さ，価格性能比を考慮してのことである．ロボットは十分動作が確認されてから，自立化しても遅くはなく，またPCを使用していれば，互換性のある組込コンピュータにシステムを移行するのみで，制御系の自立化が容易であるとの思惑もある．

ロボット制御に高性能PCを用いるにあたり，すでに，汎用のLinux[1)]を用いて，容易に安全に制御環境を構築する手法については報告した[2)3)] (本報告でも簡単に触れる)．本手法はすでに２脚歩行ロボット[4)]で，その有効性が実証済である．しかし，この段階では，ロボットのハードウェアとのインターフェイスは旧形のコンピュータに依存し

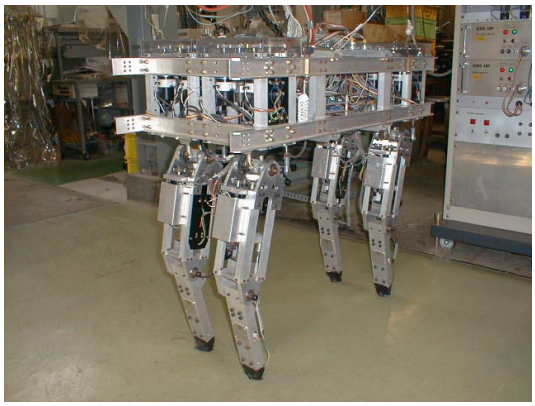


Fig. 1　４脚歩行ロボット

ており，完全な移行とはなっていなかった．そこで，次段階として，ロボットとのインターフェイス技術を開発し，高性能PCのみで完結したシステムの構築を開始した．

この計画の制御対象は，制御環境の老朽化が進んでいた，著者らの研究室の４脚歩行ロボット(**Fig 1**)とした．これまで２台の旧形PCによって制御されていたものを，１台のマルチプロセサ(SMP)PCによる制御システムに置き換え，今後の画像処理等を含む研究に備える．そのために課題となったのが，インターフェイスの確保である．４脚歩行ロボットは，12個のDCサーボモータにより駆動され，また多数のセンサを搭載している．これらの制御，状態取得のためには多チャネルのインターフェイスが必要であるが，市販品にはそれほど多チャネルのものはなく，少チャネルのものを数枚使用する必要がある．この課題は従来からあったが，以前はインターフェイス回路の自作で克服してきた．現在は拡張バスがPCIであり，一般には自作に向かないとされる．また，自作するにしても，毎回仕様の異なる回路を設計し，製作することは負担である．そこで，プログラマブルロジックを搭載した“ユニバーサルインターフェイスボード”を開発し，用途に合わせて回路をソフト的に変更することを考えた．

本報告では，このボードの紹介と，汎用Linuxを

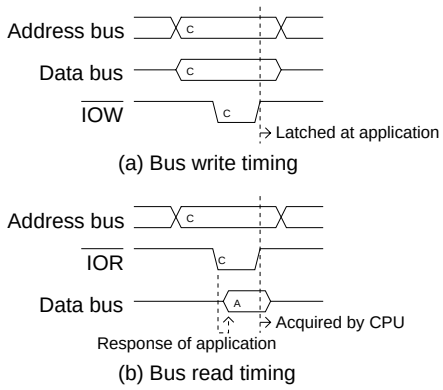


Fig. 2　従来のバスタイミング例

ロボット制御に用いる手法の応用性の４脚ロボットによる確認を行う．

2.　ユニバーサルインターフェイス

2.1　概要

コンピュータでロボット等を制御するには，インターフェイス(以下 IF)が必要である．ディジタルのパラレル，シリアル入出力，A/D変換，D/A変換などの汎用のIF，特定の機器向けの専用IFなどである．数社のメーカが汎用IFの製品群をそろえ，多くの場合はこれらを使用することで入出力可能である．しかし，これらでは不十分な場合もある．必要な機能を持つ製品が存在しない場合，また，非常に高価な場合などである．パラレルIFを使用し，その先に必要な回路を製作することも可能であるが，拡張バスに直結可能なデバイスが存在する場合もあり，IFボードそのものを製作したほうが速度的に大幅に有利な場合がある．

数年前は主流であった Cバス(NEC PC9800シリーズの拡張バス)，ISAバス(IBM PC/AT互換機の拡張バス)は **Fig 2** に示すような，ごく単純な仕様のものであった．すなわち，PC側からデータの出力を行う場合，CPUがアドレス，データバスを適切に設定した上で，IOW信号を一旦ローレベルに落とし，入力を行う場合には，アドレスバスを設定した上でIOR信号をローレベルに落とし，IF

側にデータバスを駆動させ，これを読み取る．このため，アドレスデコーダと1個のラッチICで出力ポートが，1個の3-stateのバッファICで入力ポートが容易に作成可能であった[1]．

これに対して，近年主流のPCIバスは非常に複雑なものとなっている[5]．アドレス，データ共用のバスと多数の信号線があり，その動作速度も33 [MHz]と非常に高速である．また，従来は手動であったらアドレス設定等が自動化されたため，ユーザには便利であるが，要求される回路は非常に複雑なものとなった．それでも，動作の高速性，広大なI/O空間を得やすいなどの利点があり，また，近年のPCではISAバスがほとんど搭載されないなどの問題もあり，IFの主流もPCIへ移行せざるを得ない．最近ではメーカのPCIへの移行も進み，多様なボードを入手可能であるが，時には自作の要請もある．そこで，PCI拡張IFボードの試作を行うことにした．

検討の結果，専用ICを併用することで比較的容易に製作可能であることは確認されたが，用途に応じて回路，基板を設計製作し，コンピュータに実装して動作確認をすることは，多くの手間を要し，また動作確認のためのコンピュータを危険にさらすことになる．そこで，CPLDを採用し，必要なロジック回路をその中に構築するIFボードを製作することにした．ボードを挿入してコンピュータを起動し，その後ソフトウェアによりロジック回路を設定する．ボードは1種のみで，必要に応じて回路を設定するため，開発の手間が低減され，また，必要な時に必要な回路を容易に得ることが可能となる．このボードをユニバーサルインターフェイスボード (以下 UnivIF)と呼ぶ．

なお，同様にプログラマブルロジックを搭載した類似の構造をもつPCIボードの開発キットが存在する．それらが開発キットとして使用されることを前提に，ツール類も付属するなど高価であるのに対し，UnivIFはこれ自体がボードとしては完成品であり，IFとして使用可能な設計となっていることが特徴である．

[1] これはIntel系の仕様であるが，Motorola系も大きな違いがあるわけではない

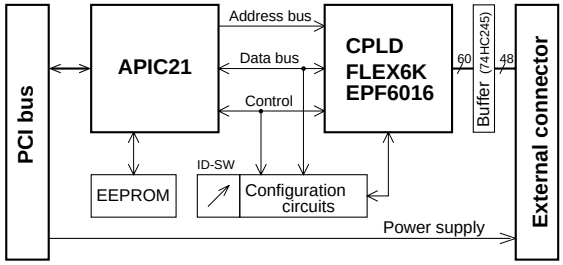


Fig. 3　ユニバーサルインターフェイスボードの構成

2.2　構造設計

2.2.1　基本構造

本ボードの概要を**Fig 3**に示す．PCIバスとのインターフェイスは専用のブリッジICである APIC21 (アドテックシステムサイエンス社製)[6][7]を使用した．市販のPCI基板開発セットには，この部分も含めてCPLDに頼るものが多いが，本ボードの使用形態を考慮すると，PCIバスへの応答は専用ICを採用することとした．これは，内部回路を容易に設定できるように，データのPC側からのダウンロードを行うことにしたため，最低限PCIバスとの接続は確保する必要があるためである．

このAPIC21のローカルバス(アプリケーション側)の配線をすべてCPLDに接続することで，ローカルバスの必要な機能を選択して使用できるようにした．また，計48 [bits]の外部との入出力信号は，6個の8 [bits]双方向バッファを挿入して CPLD に接続した．このバッファは保護用のものであり，方向はCPLD側から制御できる．さらに，PC側からソフトウェアによってCPLDの内部回路を設定するためのCPLDの設定用の回路(Configuration circuit)を用意した．なお，ID-SWは複数枚のボードを利用する場合の識別番号設定用である．

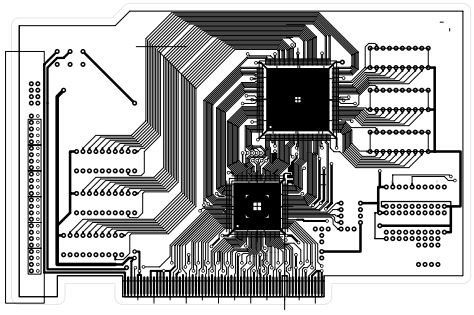
2.2.2 バスブリッジ APIC21

本ボードの中核となるLSIの一方がAPIC21である．このAPIC21はバスブリッジと呼ばれ，PCIバスと接続するために必要な機能をもち，PCIバス側からのアクセスを，**Fig 2**に示したような，容易に使用可能な形式のローカルバスのアクセスに変換する機能を持つ．すなわち，APIC21をPCIバスに直結し，ローカルバスに従来と同様な設計の回路を接続すれば，容易にPCI用のIFボードを製作できる．メモリ，I/O空間を持つほか，割り込みなども使用可能である．ただし，PCIのアクセス速度を有効活用できるよう，ローカルバスのアクセス速度も速くなっている(ウェイトは挿入可能)．今回の回路では，TTLのラッチとバッファによってCPLD設定回路を構成する他は，CPLD内部にすべて回路を構築するため，速度的な問題はない．

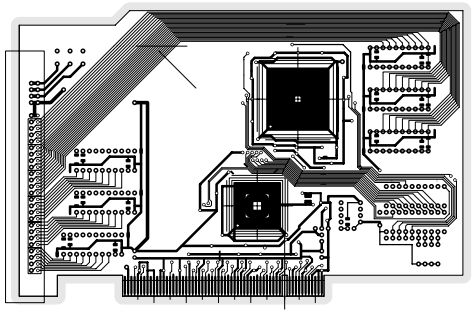
PCIバスからみた APIC21 は最大で128 [Kbytes]のメモリ空間と，256 [bytes]のI/O空間を持つデバイスである．この空間は，PCの起動時に適当な場所に設定されるので，それを検索してアクセスすることになる．なお，PCIデバイスに固有のベンダIDは 0x136c，デバイスIDは，本ボードには0xa026が割り当てられている．

2.2.3 CPLD FLEX 6000 シリーズ

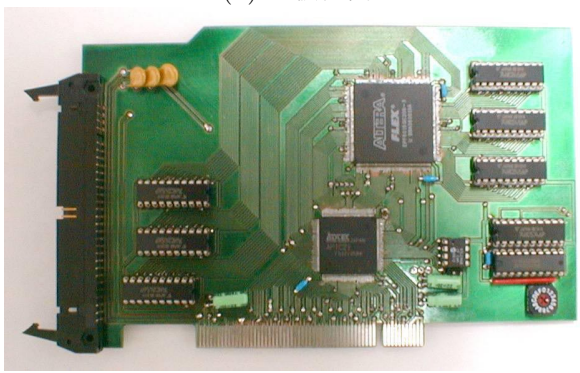
本ボードは大規模プログラマブルロジックを搭載することによって，IFに必要な回路を自由に設定，変更できるようにした．今回採用したのは アルテラ社のCPLD，FLEX 6000 シリーズEPF6016である[8]．これは，1ビットのラッチと簡単なロジックを含むロジックエレメントが1320個含まれるデバイスであり，入出力が171本ある208ピンQFP パッケージのものを使用した．後述するロボットの制御システムのIFを設計して，24 [bits]の2相エンコーダカウンタと8 [bits]PWM出力を各8チャネル組み込んで余裕が多少ある程度である．



(a) 基板表面



(b) 基板裏面



(c) 基板表面実物写真

Fig. 4 ユニバーサルインターフェイスボードの実装

このCPLDの内部回路の設計にはアルテラ社が無償提供している MAX+plusII baseline を使用した．基板設計にあわせてCPLDのピンの機能を設定し，内部のロジック回路を設計し，CPLD設定データを出力する．このデータをソフトウェアによってPCに搭載したUnivIFのCPLDに書き込むことで，ボードが動作する．設計した回路は事前にシミュレーションも行うことが可能であり，確実に動作するIFを容易に得ることができる．

デバイスの詳細についてはメーカの資料を参照して頂きたい．

2.3 実装と評価

以上の検討をもとに，回路を設計し，基板(両面2層)を製作し，部品を実装した．回路，基板の設計と言っても，ただ平行に配線するのみであるが，配線総数は300本を超えるものとなる．外部との接続は60 [pin]のコネクタを使用し，48本の入出力信号線のほか，簡単な外部回路を接続するのに便利なよう，電源の供給も行っている．実装した基板を**Fig 4**に示す．図中中央に見えるのが中核のLSIで，上がCPLD，下がAPIC21である．

このボードはただ製作しただけでは，何ら機能を持たない．CPLDの設定も必要であるが，APIC21の動作設定も必要である．APIC21に関してMS-DOS用の設定プログラムは提供されたが，主な使用環境がLinuxであるため，APIC21のデータシートをもとに，Linux用の設定プログラムを製作した．また，CPLD用の設定プログラムもデータシートや参考プログラム[9]をもとに製作し，Linux上で可能とした(Windows用のCPLD設定プログラムは開発中)．

現在，ボードは3枚試作し，うち2枚を４脚ロボットの制御用PCに搭載して，ロボットのDCサーボモータ制御用に使用している．１枚のボードに8チャネルの24 [bits]の2相エンコーダカウンタ(4逓倍)と8チャネルのPWM発生回路を構成し，8個のモータの制御を行う．PWM出力はボードより供給する正負電源を利用して，サーボアンプ入力のアナログ電圧に変換している．

ボードの１枚のコストを算出したところ，基板加工に要する消耗品費を含め，12,000円程度であった．人件費等を含まないという条件はあるものの，市販のボードで必要な機能を得るのに比較し，非常に低コストである(市販ボードは必要以上に高機能なことが多くコスト高となる原因になっている)．また，ボードを在庫しておくことで必要なIFをすぐに構成でき，使用後の再利用も容易であるなど，コストパフォーマンスは良いと考えられる．また，回路の手直しも容易で，今回の用途でも，信号ノイズに起因する誤動作をCPLDの回路設計にディジタルフィルタを追加することで対処している．

今回はロボットの制御用としてボードの開発を行ったが，用途は限定されるものではない．今後は容量の大きなCPLDに変更して(ピンが互換なのでCPLDを変更するのみ)，ディジタル信号処理などに応用していく予定である．

3. Linuxによる制御環境の構築

3.1 背　　景

これまでロボット制御用OSとしてMS-DOSを用いて来た．MS-DOSはシングルタスクの単純なOSであり，一旦ユーザのプログラムが起動後はコンピュータ全体をユーザが自由に操作可能であり，制御を行うには好都合である．しかしながら，MS-DOSでは今日のコンピュータの性能を生かしきれず，他のOSへの移行が必要であった．

MS-DOSがシングルタスクのOSであったのに対して，現在はマルチタスクのOSが主流である．マルチタスクのOSは，実行すべきプログラムを同時に複数持ち，それらを交互にCPUで実行することで，擬似的に並列実行しているかのように見せる．一般用途向のマルチタスクOSにおいては，原則として各プログラムは平等に扱われる．そのため，個々のプログラムがいつの時点でCPUによって実行されるかが不明であり，ロボット等の制御には向かないとされてきた．ロボット等の制御にはリアルタイムOSと称する，規定の時刻に目的のプログラムを実行可能な機能が組み込まれたOSが一般には使用される．また，通常のマルチタスクOSでは，１つのプログラムの動作が原因でシステム全体に問題が生じないように，ハードウェアに対する直接的なアクセスが制限されているのに対し，これらリアルタイムOSでは自由なアクセスが

可能となっている点も，異なる．

さて，以上のような背景の下，従来MS-DOSで行っていた制御を移行するため，リアルタイムOSの一つであるRealTime-Linux(RT-Linux)[10)11)]の採用を検討した．すでに，ロボット制御以外の用途でLinuxを使っていたため，それと互換性を有するRT-Linuxを採用することで，習得期間の短縮を狙ったものである．しかしながら，検討を進めるうちに，RT-Linuxの欠点が見えてきた．それは，制御ルーチンの実行の際にメモリ保護機能が働かないことである．プログラムを作成する場合，I/Oアドレスを誤りシステムに打撃を与えることは稀であるが，メモリの場合にはアドレス(C言語のポインタ)を誤ることで，他のプログラムが使用している領域を誤操作する危険性は高い．一般のマルチタスクOSでは，このような場合にはプログラムの実行を停止させ，他のプログラムを保護するが，RT-Linuxはその機能がない．このことは，プログラムの誤りによって，OS毎コンピュータが停止する可能性を意味し，最悪の場合，データ類の破壊も招く．また，プログラムの誤動作の原因を突き止めることも困難である．

そこで，制御用に使用するOSとして，汎用のLinuxを使用することを考えた．別の目的でLinuxのソースを解析していた際に，偶然に，汎用のLinuxでも制御に必要な動作をすることが可能である，ということに気付いた．その原理に基づいて基礎実験を行うことで，ロボットの制御に使用可能であることを確認し，まず２脚歩行ロボットの制御システムの移行を行った．OSを汎用のLinuxとすることで，メモリ保護機能の下，安全に研究開発を行うことが可能となり，また，OSのマルチタスク機能を利用することでマルチプロセス(複数のプログラム)の並列協調動作による制御システムを構築することに成功し，研究効率の向上を達成した．今回は，その成果をもとに，４脚ロボットの制御システムのLinuxへの移行を進め，本手法の有効性の追試を行うこととした．以下に本手法の原理を解説し，システムの設計について述べる．

なお，RT-Linuxの弱点を克服すべく，別途開発されたART-Linux[12)13)]も存在する．これはメモリ保護を行いつつ，リアルタイム処理を可能とする点で理想的なOSといえるが，現時点では十分な安定性を確保することが難しいと言われている．汎用Linuxを用いる本手法は，ART-Linuxとプログラムの作成手順が近いことから，将来的な移行も選択肢の１つであると考えている．より高精度な実行のため，ART-Linuxの早期の完成が望まれる．

3.2 汎用 Linuxによる一定周期実行

すでに，各所で本手法については述べているので，ここでは簡単に述べることにする．

ロボット等を制御する場合，二つの機能が必要と考えられる．一つはハードウェアに対するアクセスであり，Linuxにおいては，管理者権限があれば可能である．もう一方は一定周期で制御則を実行可能であることである．これは一般的な制御則は一定周期で制御演算を行うことを前提としているためである．本節では後者の手法を述べる．

マルチタスクOSでは，実際に実行するプロセス(プログラムの実行単位)を切り替えるため，スケジューリングと呼ばれる動作がたびたび行われる．この際，OSは多数あるプロセスの中から，ある条件に従って一つを選び，CPUによる実行を開始する．Linuxにおいて，この条件は各プロセスの持ち時間の大きさである．Linuxが利用している一定周期のタイマ割り込みの時点で実行されていると，持ち時間が減じられる．その直後にもスケジューリングも行われるため，連続して実行中のプロセスは，徐々に持ち時間を減らし，他のプロセスにCPUを譲ることになる．

このタイマ割り込みでは，Linuxの時間に関する処理を行っているが，その中に，休眠プロセスを

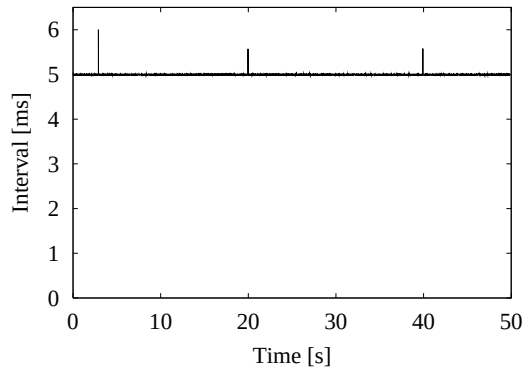


Fig. 5　5 [ms] 周期実行の精度

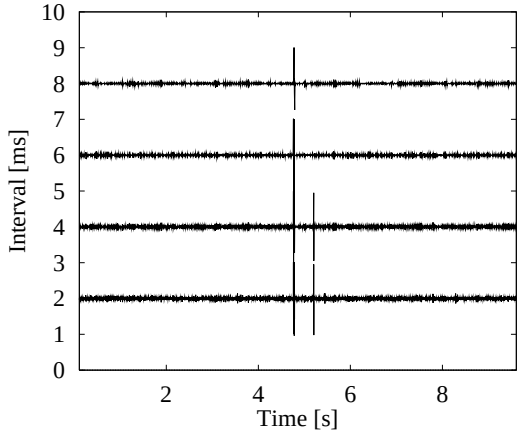


Fig. 6　複数周期の実行

起こすという作業がある．一般にLinux上では数十のプロセスが実行されているが，その多くはデバイスの入力待ちや期間を指定しての休眠状態にある．この休眠状態にあるプロセスはスケジューリングでは考慮されない．さて，あるプロセスが時間を指定して休眠していたとする．このプロセスが実行可能な状態に戻るのはタイマ割り込みの段階である．この直後にスケジューリングが行われるが，起こされたプロセスが十分な持ち時間を有していた場合はこのプロセスがCPUで実行されることになる．これが本手法の原理である．

まず，周期実行したいプロセスを，適切な時間を指定して休眠させる．すると，時間経過後最初のタイマ割り込みで実行可能な状態となり，持ち時間が十分であればそのままCPUで実行される．必要な処理を行った後，再度休眠に入り，同じことを繰り返す．必然的にこのプロセスは休眠時間が長くなるため，持ち時間をあまり消費せず，休眠終了後に実行される確率は高い．このため，Linuxのもつタイマ割り込みの精度に近い精度で周期的に実行されることが期待される．ただし，実際に制御に使用するには多少の工夫が必要である．タイマ割り込み間隔は標準で10 [ms]であるため，周期を10 [ms]単位より細かく設定できない．細かく設定する場合には，これを1 [ms]などに変更する必要がある．また，周期安定性向上のため，プロセスの優先度を高める必要もある．

実際の周期実行例を **Fig 5**に示す．これは 5 [ms] を目標として１万周期実行したものである．本手法は あくまでプロセスとして周期実行を行うため，稀にLinux本体(カーネル)の処理の影響を受け周期が乱れるが，それ以外は高精度に周期が保たれる．よほど高精度な目的でない限り，周期が稀に変動することは制御にはほとんど影響しないと考えられる．特にロボットの場合は，それ以外の外乱の方が遥かに多い．この例は1プロセスで単一周期の実行を行ったものであるが，別途特殊なドライバを使用することで，**Fig 6**に示すように単一プロセスで複数周期を得ることも可能である．

これらの具体的な手法は一般公開しており[3)]，詳細についてはそちらを参照して頂きたい．

3.3　マルチプロセスによる制御

Linuxへの移行を行った最大の理由は，PCの性能を有効活用するためであった．そのため，従来MS-DOSで製作していたような，単一プログラムによる制御を行うことも選択肢の一つであった．しかし，制御システムの規模が大きくなるにつれ，単一のプログラムで作成した場合に，処理の順序などを検討する必要が出てくる．そこで，Linuxのマルチタスク機能を利用し，処理ごとに別プロセスとし，それぞれに優先順位を設定することで，処理順序決定をOSに依頼することとした．さらに，ロボットの制御に必要なルーチンを部品化するこ

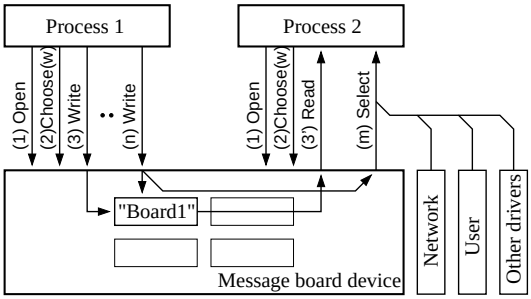


Fig. 7　メッセージボードデバイス

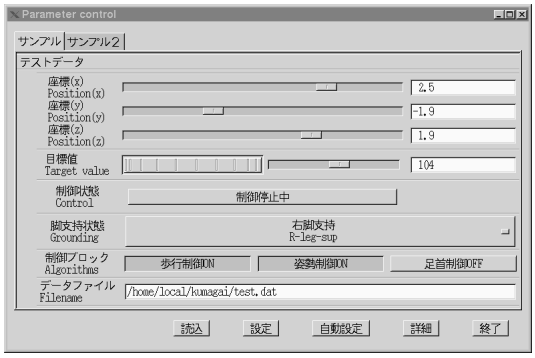


Fig. 8　VariableSet 操作監視 GUIパネル

とで，システムの構成変更を容易とした．複数のプロセスで処理を行うためには，プロセス間の通信機構が必要である．Linuxには標準で複数種の通信機構が存在するが，適当なものがなかったため，新たに製作した．本節ではそれらについて述べる．

3.3.1　メッセージボード デバイス

まず，プロセス間のデータ交換を可能とするメッセージボードデバイス(MSGB)について述べる．マルチプロセスで制御する場合には，いくつかの要求がある．制御を行うためには過去のデータ履歴より現在のデータが必要であり，また情報の分岐が必要なため，FIFO型の手法は好ましくない．また，CPUを無駄に使用しないために，ロボットの制御に必要な部品となるプロセスは，必要となるまで休眠状態におき，データの入力と共に演算を開始させるべきであるため，データの伝達とともに，そのデータを待機しているプロセスを起こす機構が必要である．さらに，他のプロセスによる監視を可能にすると，制御のモニタが容易となり便利である．

このような要求に答える通信機構として，MSGBを開発した．これは伝言板のようなものであり，複数の名前をつけた領域からなる(**Fig 7**)．情報提供側のプロセスは独占した書き込み権を持ち，データの書き込む．この領域は任意のプロセスが読み取ることができ，データの更新を通知してもらうよう，休眠待機することも可能である．この通知には，Linux(UNIX)ではデバイスの待機に標準的に用いられるselectを用いたため，MSGBと同時に，ネットワークからの入力やユーザからのキー入力などを待機することが容易である．

3.3.2　VariableSet

MSGBは任意のデータを通信可能である．このような場合，C言語の構造体によってデータセットを定義し，送受する手法がよく用いられる．このデータを制御には直接関係しないプロセスで読み取って，制御データの記録をすることも可能であるが，このデータの解析を行うには，制御プログラム内で定義したデータ形式を把握しなければならない．また，一つ制御データを増やしたのみで，関連する全プログラムを変更しなければならなくなる．

これらの問題を解決するために，VariableSetと呼ぶ形式を定義した．これは，構造体のようなデータの配列領域と，このデータの定義領域からなる．データの定義領域には，個々のデータの識別名，種類(整数・浮動小数・文字列)および大きさ(配列要素)，構造体上のオフセットが記載されている．このため，識別名をキーとして，データの抽出が容易となる．さらに，この定義情報をデータの後方に配置することで，初期化時のみ定義情報を処理し，2回目以降はデータ領域のみを読み書きする

ことで，オーバーヘッドは最小限にとどめるようにした．

形式を共通化したことで，各種ツールを汎用化することができた．開発したツールは，GUIによる監視・操作ツール(**Fig 8**)，複数のVaribleSetを同時に記録可能なツール，ネットワーク伝達ツールなどである．これらにより，ロボットの制御データをVariableSetでやり取りする限りは，専用の操作部分や，記録用のコードを書く必要がなくなり，制御手法本体のみに集中することが可能となるため，生産性が向上した．

3.4 制御階層設計

複数のプロセスに処理を分割する際には注意が必要である．あまり細かく分割すると通信やプロセス切替のオーバーヘッドが増加する．逆にあまり機能を大きくすると，汎用性が低下する．そこで，２脚歩行ロボットの制御系を分割する場合には，主たる制御部分を運動学演算プロセスと主制御プロセスに分割し，その他に制御用の部品として，画像処理プロセスやトレッドミルの制御プロセスなどを用意することとした．後者の分割は当然に思えるが，制御に必須である運動学演算を分離したのにも理由がある．運動学演算は，多くの制御で利用するためライブラリとしての性格が強く，また，これ単体で稼働させ，指令値をGUIツールから与えることで，ロボットの手動操作も可能である．さらに，ロボットの機構パラメータを多少修正した場合に，運動学演算プロセスのみ修正して，主制御プロセスは無修整で済ますことが可能である．一体のプログラムの場合，すべての実験用プログラムを再構築する必要が生じる．

この２脚ロボットの分割は成功であったと考えている．制御システムが完成後に新たに腕が追加された際にも，腕の運動学演算プロセスを部品として追加している．そこで，４脚ロボットの制御システムを構築する際にも，同様の分割を行うこととした．ただし，２脚ロボットとは大きく異なる部分がある．既存の２脚ロボットの制御システムと，現在構築中の４脚ロボットの制御システムの比較を **Fig 9**に示す．２脚ロボットの制御システムがハードウェア関連の処理を別のコンピュータに委託しているのに対し，今回はすべて１台のコンピュータで賄うこととした点である．そこで，モータのサーボ制御を行うプロセスと，センサの情報処理を行うプロセスを別途追加することとした．これら二つはロボットに電源を入れておく限り稼働させるものであり，常にロボットの状態を監視し，制御する部分である．特にセンサの情報処理は，角速度センサの積分を継続して行う必要があるなど，連続稼働が求められる．それに対して，運動学演算，主制御プログラムは実験実行時に機能すればよいので，実行停止は任意となる．

3.5 制御システムの実装

現在，ロボットの制御システムの実装は進行中である．UnivIF 上に構築したIF回路の動作を確認し，2 [ms]周期のモータのPIDサーボプロセスが完成し，また，電源投入時に必要な，関節原点の決定プログラムの移植まで完了している．現状では，各モータの回転角の指令・制御に問題がないことが確認されたところであり，運動学演算プロセスの開発を進めている．今回新たに試みた部分である，汎用Linuxによるハードウェアの直接制御部分は達成したことになり，今後は２脚ロボットと同様にシステムを構築するのみであるため，重大な問題は発生しないと考えられる．

今後の予定としては，運動学演算プロセスを完成して，脚関節のフィードフォワードのみによる歩行の動作確認をし，センサ情報の処理プロセスを実装することで，移植作業の完了とする．これにより，移植前以上の実験を行うことが可能とな

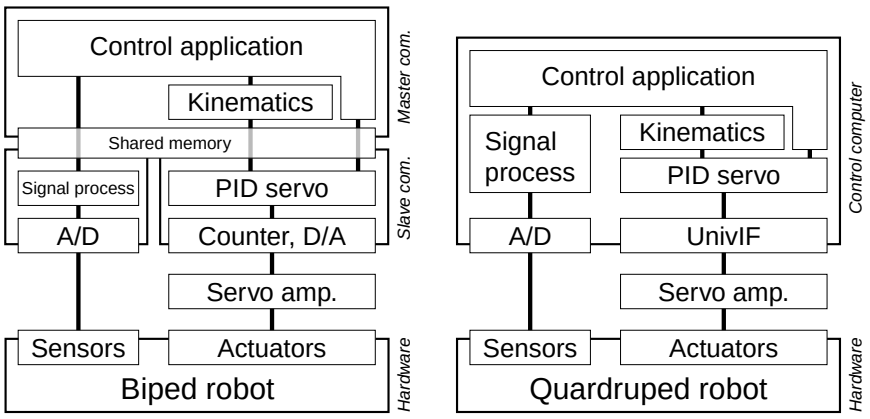


Fig. 9　２脚ロボットと４脚ロボットの制御システム階層

り，研究効率が向上することが期待される．

4.　おわりに

本発表では，４脚歩行ロボットを題材として，現在主流の安価で高性能なPCで，ロボット制御を行うための環境開発手法について紹介を行った．

- 主流となった，高速な拡張バスであるPCIバスに適合する自作インターフェイスが製作可能であることを示し，その回路部分をCPLD内部に動的に構築する“ユニバーサルインターフェイスボード”の開発を行った．これにより，低コストに，必要とされる機能をもつインターフェイス得ることが容易となった．
- ロボットの制御システムを汎用のLinuxで構築する手法の紹介を行った．この手法は，設定や習得のコストが低いばかりか，開発時の安全性にも優れ，研究効率の向上が期待できる．また，モータのサーボ制御にも問題なく使用できることが確認された．完璧な性能は得られないものの，ロボットの制御に十分な性能は得られる手法であると言える．

今回紹介した内容の一部は，すでにWWWにて一般公開している．ユニバーサルインターフェイスボードに関する情報も，順次公開し，量産化を目指したいと考えている．

参考文献

1) “The Linux Home Page”, http://www.linux.org/
2) 熊谷正朗, 江村超: “汎用Linuxによるロボット制御手法の提案　－第1報 非RT Linuxによる定周期動作の実現－　－第2報 マルチプロセスによる２脚歩行ロボットの制御の実際－” 第17回 日本ロボット学会学術講演会, 講演番号 3C33 (1999), 847–850.
3) 熊谷正朗:　“Linuxでロボット・ハード・制御”, http://www.mechatronics.mech.tohoku.ac.jp/~kumagai/linux/
4) 甲斐田壮，熊谷正朗，江村超: “腕を有する２脚ロボットの制御に関する研究”, 計測自動制御学会東北支部第192回研究集会 資料番号 192-3
5) 吉田幸作他: “トランジスタ技術SPECIAL No.65　PCIバスの基礎と応用”, CQ出版社(1999)
6) アドテックシステムサイエンス:　“PCIターゲットインターフェースアダプタ　APIC21”, http://www.adtek.co.jp/apic21.html
7) 中島稔: “PCI-汎用バス・ブリッジ・カードの製作”, CQ 出版社 トランジスタ技術 2000年12月号
8) ALTERA社:　“FLEX 6000 Programmable Logic Device Family Datasheet”, http://www.altera.com/
9) 山本隆司: “Autonomous Agent Farm”, http://www.aafarm.com/
10) “Real-Time Linux”, http://www.rtlinux.org/
11) 船木陸議, 羅正華: “LINUX リアルタイム計測/制御 開発ガイドブック”, 秀和システム, 1999.
12) 石綿 陽一, 松井 俊浩, 国吉 康夫: “高度な実時間処理機能を持つLinuxの開発”, 第１６回日本ロボット学会学術講演会予稿集, 355–356, 1998.
13) 石綿 陽一: “ART-Linux誕生の経緯と使い方”, “リアルタイム制御を実現するART-Linuxの設計と実装”
柴田智広: “ART-Linuxによるリアルタイム処理への適用と応用”,
インターフェース, 1999年11月号, CQ出版社, 1999.