データフロープロセッサ：D³Pの設計思想

THE DESIGN CONSIDERATION OF DATA FLOW PROCESSOR: D³P

安原 宏　伊藤徳義　瀬賀明雄　上原一矩
Hiroshi YASUHARA　Noriyoshi ITO　Akio SEGA　Kazunori UEHARA
沖電気 研究所　プロセッサ研究部
OKI Electric Industry Research Lab. Processor Dept.

I. はじめに

現在，様々のタイプのデータフローマシンが開発されている[1][5][6][8][9]。MITのDennis他により提案されているマシン[2]は，命令レベルの高並列処理を指向しており，命令メモリと演算装置の間を多くの通信線で結合している。しかし，この接続にはserial lineを用いており，パケットあたりの通信遅延はかなり大きくなっている[4]。従って，並列度のきわめて大きいアプリケーションに向いている。

一方，主としてプロシージャレベルの並列性をねらっているマシンも提案されている[5][10]。プログラムを，比較的シーケンシャルに実行される処理単位（タスク）に分割して，各タスクをそれぞれPMU（プロセッサ－メモリ・ユニット）に割当てる方式である。しかし，このような方式は並列タスクへの分割が問題となる。すなわち，プログラマ（又はコンパイラ）は独立に動作するタスクを意識してプログラムを分割しなければならない。

本論文ではタスクレベル並列実行はもちろん，1つのタスク内でも命令の並列実行が可能なデータフロープロセッサ D³P（Distributed Data Driven Processor）を提案する。

II. システム構成

2.1 概要

データフロープログラムは，複数のノードと各ノード間を接続する方向性のあるリンクで構成されたデータフローグラフで表現される。グラフ中の各ノードは実行すべき処理機能（function）を示し，リンクはノード間のデータの流れを示している。

各ノードは，その全ての入力リンクにデータ（トークン）が現われたときに実行可能となり，それらのトークンを取込み，指定されたfunction を実行し，結果をノードの出力リンクに出力する。

このようなデータフローグラフにおいては，グラフ内の各ノードの実行はその入力データの到着にのみ依存しているので，本質的に非同期であり，複数のノードが同時に実行されるという可能性が生ずる。

一般的に，ノードの処理単位を細分化すればする程，並列実行の可能性は増加するが，しかし各ノード間のリンクの数が増し，従って通信コストが高くなる傾向にある。

本論文では，各ノードで示されるfunctionを2つのタイプに区別している。1つは処理装置で直接インタプリトされるprimitive functionであり，今1つはnon-primitive functionである。non-primitive function は，primitive function又はnon-primitive functionをノードとするデータフローグラフで示される。（以下，特にこと

めらない限り，前者を primitive function，後者を単に function と呼ぶ。）

D^3P は，処理装置と命令メモリが比較的密に結合された DFE（Data Flow processing Element）を，多数結合したアーキテクチャを採用している。このような方式をとることにより，primitive function（命令）レベルはもちろん，function（タスク）レベルの2つのレベルの並列実行を実現することができる。

DFE は，最終的には1つないしは少数の LSI チップ内に収容することをねらっており，システム全体のパフォーマンス／コストの大幅な向上をねらっている。

2.2 物理的構成

基本プロセッサエレメントとして，前述の DFE のほか，LCP（Local Control Processor）及び CP（Control Processor 又は HOST Processor）があり，1つの LCP と複数の DFE で1つの DFU（Data Flow Processing Unit）が構成される。DFU は外部インタフェースを介して C-bus 上に結合されている（図2.1）。CP は，DFU 群を一括管理すると共に，objects の管理も行う。

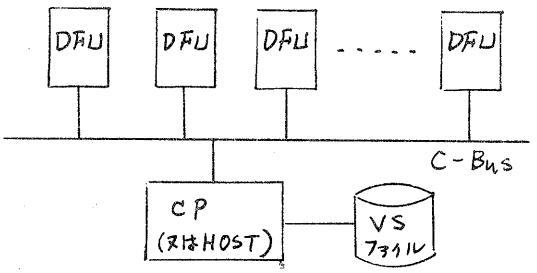


図 2.1 D^3P システム構成

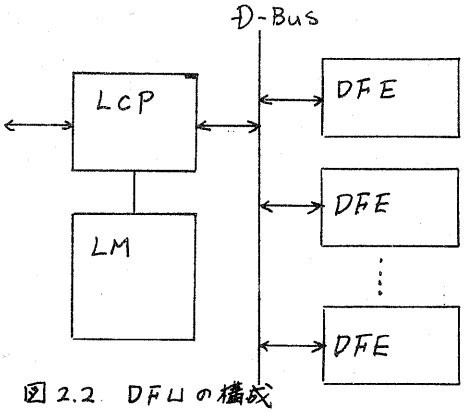


図 2.2 DFU の構成

2.3 論理的構成

D^3P システムで管理される object としては function と data segment があり，論理的にまとまったいくつかの function が集まって cluster を構成する。function 及び data segment は，仮想空間（VS）内に写像され，システム全体でユニークな identifier（仮想アドレス）が与えられる。function 内のアドレスは相対アドレス（ローカルアドレス）である。

2.4 object の割当て

各 object は，VS の中で個有のアドレスが付与され，1 DFU には 1 Cluster が割当てられ，Cluster 中の各 function は1つの DFE に割当てられる。

Ⅲ. リソース管理

3.1 ハードウェアリソースの管理

CP は DFU 群の Busy／Available 状態を監視しており，Busy の場合はどの Cluster が割当てられているかの情報を保持している。各 DFU 内の DFE 群は LCP が管理しており，CP の場合と同様に，DFE の Busy／Available 状態を監視している。DFE が Busy のときは，どの function を実行中であるかを保持している。

3.2 Objects の LM へのロード

各 object は 2.3 で述べたように VS 上の空間内に写像されるが，システムが処理動作中の場合，その一部は DFU 内の LM（Local Memory）内に存在する。実行時に必要な object が LM に存在しなければ，LCP は CP にデータ要求を出す。LM 内の空間配置は，図3.1 に示す。

3.3 function のDFEへの割当て

各functionは実行可能になるとfree DFEの1つに割当てられ，実行される。この割当て管理プログラムはLM内のRM領域に常駐である。LMのFDT領域には，DFEにClusterが割当てられたとき，そのCluster内で使用される全functionの属性がロードされる。functionの属性には，プログラムの格納されているディスクアドレス，プログラムサイズ，あるいはデータ空間サイズ等の情報が含まれる。

functionが起動されたとき，その起動要求に対して付与されたアクティベーション番号（ACT#），functionの実行結果を返すべき親functionへのポインタ，及び割当てられたデータ空間の仮想アドレス等はFCS内のエントリに退避される。FCS内のエントリはfunction起動時に割当てられ，functionの終了時に開放される。FI領域にはCluster内で使用するfunctionのプログラム原本の一部又は全部がロードされている。

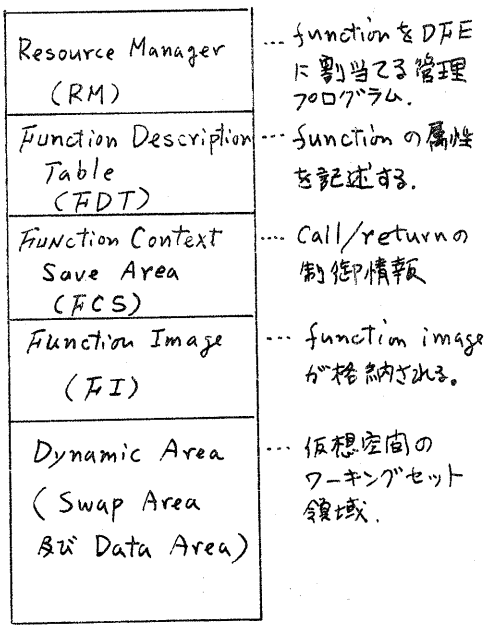


図3.1 LMの配置

Dynamic Area は，仮想空間のうち，実メモリ（LM）にロードされているページの集合（ワーキング・セット）である。この領域にない仮想ページはVSファイル内にページアウトされているか，又は未使用ページである。仮想空間を管理するためのプログラムやページテーブル等は，前述のRM部に常駐である。この仮想空間内には，各functionで使用する構造体データを格納するためのデータ領域やスワップアウトされたfunctionのコード等が置かれる。仮想空間の割当て管理はダイナミックである。すなわち，functionが起動されたとき，仮想空間内の未使用ページがそのデータ領域として割当てられる。またスワップアウト領域の割当てもその必要性が生じたときのみ発生する。

IV. function の実行管理.

4.1 function の起動処理

或るfunctionの全入力アーギュメントがそろったとき，function call命令が実行される。このとき，function callを実行したDFEよりLCPに割込みがかかり，functionの起動要求が通知される。LCPには起動されるfunctionのFDT内のindex及びアーギュメント・リストのポインタが渡される。LCPは，まず，起動functionに割当てるためのfree DFEが存在するか否かをチェックする。free DFEの存在するときは，FDT内の属性に従って，プログラムをそのDFEにロードすると共に，仮想空間内にデータ領域を確保してアーギュメントリストの実体をコピーし，functionの先頭命令を実行可能状態にする。以降，DFEはそのプログラムを実行する。free DFEのないとき，その起動要求はwait queueにつながれる。

4.2 function の終了処理

各functionはDFE内で，データフロー的に次々に実行される。functionの結果が求められるとresult listを作成する。全結果が得られたとき，return命令を実行してLCPに通知される。LCPは，その親functionにresult listを渡したのち，その終了したfunctionが占有していたリソースを解

放する。このとき，wait状態のfunctionがあれば，解放されたDFEを割当てる。

4.3 functionのスワップアウト

function call命令以外の命令を実行中でないDFEのfunctionはスワップアウト可能であるという。スワップアウト可能となったDFEはLCPに割込みを発生し，その旨通知する。このとき，wait queueに起動要求がつながれていれば，LCPはまず，DFEのスワップアウトを行う。このとき，そのfunctionのステータス情報も退避される。次に，待ち状態にあったfunctionをロードしてDFEを起動する。スワップアウトされたfunctionは，その子functionが終了したとき，再び起動される。

V. DFEの構成

5.1 概要

DFEは図5.1のように，IM(Instruction Memory)群，IEC(Instruction Enabling Controller)群，PU(Processing Unit)群，DC(Data Cashe)，及びこれらの間を結ぶバスで構成されている。これらの装置は最終的には1チップのVLSI中に収容される。

IMはデータフロープログラムを格納するためのメモリであり，m-way interleaving方式を採用して命令フェッチのアクセスネックを解消している。

IECはIMに格納されている命令群のうち，実行可能になった命令群を監視し，IMの命令フェッチを制御する。

PUは実行可能になった命令を実行する演算装置であり，複数個用意されている。各PUは独立に動作可能である。

DCは，LCPにより管理されるデータ空間のキャッシュとして使用される。このキャッシュはもとして構造体データの処理効率を上げるために設けた。

これらの装置の間は，リング構造をしたバス群で結合されている。装置間の通信ネットワークは様々のものが考えられるが，本方式では簡単な共通バス方式を採用した。従って，DFE内での処理の並列度はあまり多くを期待できないが，D^3Pの構造をマルチDFE構成にすることによりシステム全体のパフォーマンス向上をはかっている。

5.2 IM及びIEC

IMは図5.2のようにオペコード部とオペランド部に分割されている。オペコード部の各エントリとオペランド部の各エントリは対応しており，両者のエントリを合わせて1つの命令を表わしている。PUの命令実行によって得られた結果はオペランド部に格納される。このとき2つの入力オペランドがそろえば，その命令は実行可能となる。命令が実行可能となったとき，PUの1つに送られる。PUがBusyのときは，その命令アドレスがFIFOの記憶(EIスタック：Enabled Instruction Stack)に

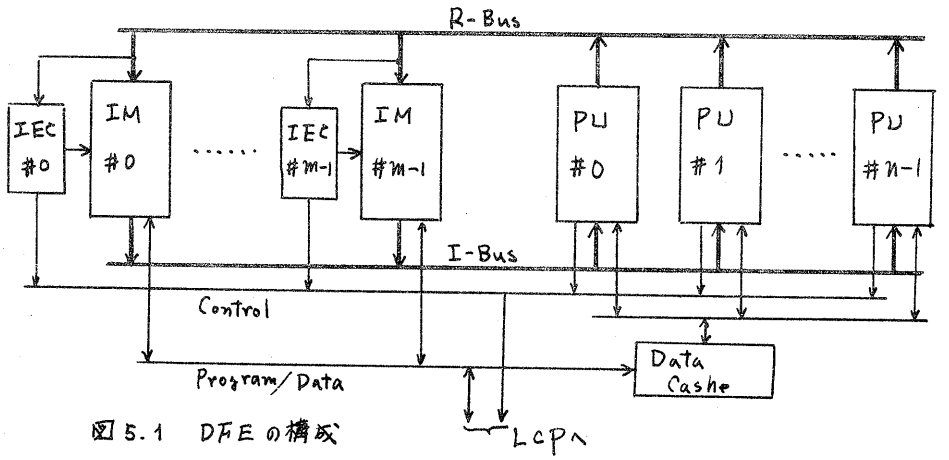


図5.1 DFEの構成

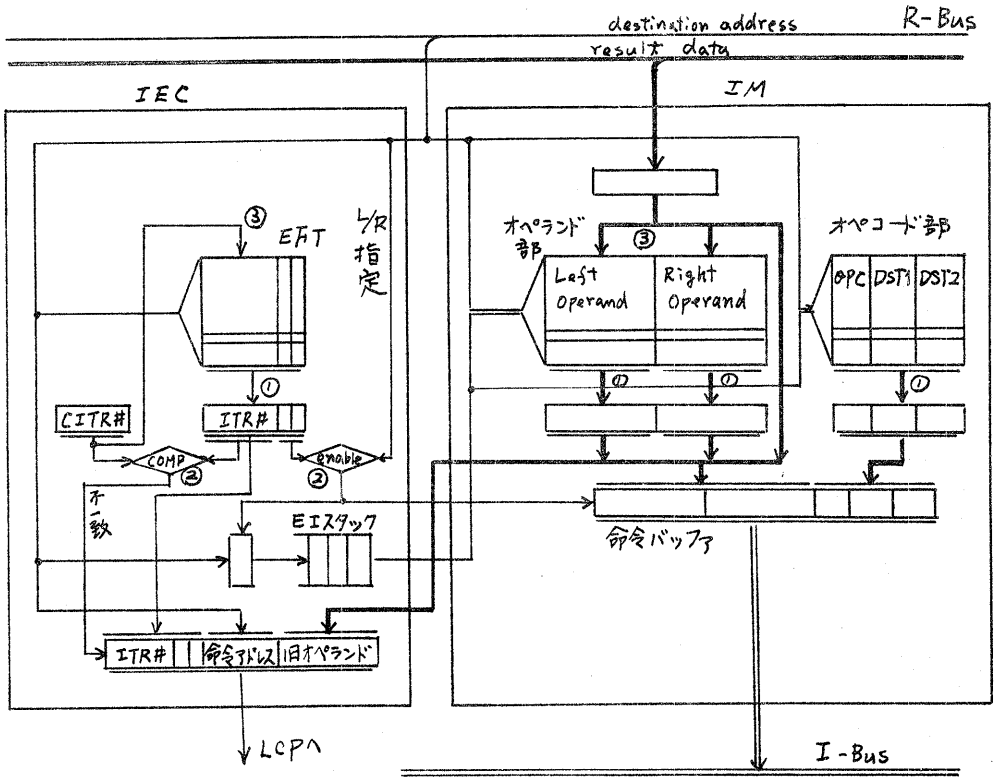


図 5.2　IMとIECの構成

一時的に格納される。EIスタックに格納された場合は，後にPUがBusyでなくなったときに命令フェッチを行う。

IECはオペランドの到着時に命令の2つのオペランドがそろったかどうかを検出する。IEC内にはIMの各命令に対応して2つのフラグが保持されている。このフラグをOEF（Operand Enable Flag）と呼び，OEFの格納されているテーブルをEFT（Enable Flag Table）と呼ぶ。EFTエントリの2つのOEFは，命令の左オペランド及び右オペランドに対応しており，いずれかのオペランド値がR-Bus（Result-Bus）より到着したときセットされる。このとき，到着したオペランドと反対側のオペランドが有効のとき（OEFがオンのとき），命令の両者のオペランドがそろったことになり，そのオペランドを含む命令は実行可能となる。

この外，EFT内にはrecursive call要求が発生したとき，残留トークンのsave/recoverを制御するための情報が格納されている。recursive function call命令が実行されると，LCPはその call要求に対して新たなiteration番号（ITR#）を与える。このITR#はfunction起動時にIEC内のCITR#レジスタにセットされる。DDPではrecursive callの際の各call要求間でIM中のコードを共有するようにした（従って，この場合IMコードのスワッピングは不要となる）。すなわち，各recursive call要求に対しては異なるITR#が与えられる。データパス（リンク）上に，スタック機能を実現するためには，後のrecursive call要求の実行中には前のfunctionの実行中に残っていたトークン（残留トークン）を一時的に退避する必要がある。IEC内には，このためのITR#比較回路，及び残留トークンを退避するための機構を設けた（図5.2参照）。

5.3 IMおよびIECの動作

IMおよびIECは，R-Busに結果が到着する毎に基本的に以下の3つのサイクルで動作する。

① read cycle
② compare cycle
③ write cycle

① read cycle: R-Busより与えられたdestination addressをラッチし，EFTをフェッチする。同時に，IMのオペコード部，オペランド部も読み出し，命令読出しレジスタにラッチする。

② compare cycle: CITR#とEFTより読出したITR#を比較する。両者が一致したときは，さらにEFTより読出したOEFTとdestination中で指定されているL/R（Left/Right Operand）とにより，該当する命令が実行可能になったかチェックする。

③ write cycle: EFTエントリに新しいOEFTおよびCITR#の内容を書込むと共にR-Busより渡されたオペランドデータをIMのオペランド部に書込む。②のサイクルで，ITR#が一致しなかったときは，①のサイクルでセットした（ITR#，命令アドレス，旧オペランド）の組をDFEの外部インタフェースを介してLCPへ転送する。

このとき，命令が実行可能であれば，I-Busへ命令を転送するが，PUがBusyの場合はその命令アドレスは一時的にEIスタックへ格納される。

5.4 命令フォーマット

IMに格納される命令は固定長（88ビット）であり，図5.3のようなフォーマットをしている。

OPCはオペコードであり，実行すべきprimitive functionのタイプを指定する。

DST1およびDST2は，命令の実行完了後に結果を転送すべきdestination addressを示している。いずれも図5.3(2)のフォーマットをしている。NIAおよびL/R指定によってデータを格納すべきオペランドアドレスが示される。SOE（Single operand Enable）指定がオンのとき，次の命令は1つのオペランドの到着によって直ちに実行可能となる（すなわち，次命令は1オペランドの到着で実行される）。NDS（No Data Store）指定がオンのときは，データをIMのオペランド部に書き込まない。このとき該当するOEFTのみがセットされる。NDS指定は例えば次命令のオペランドが定数である場合に使用される。

OPR1およびOPR2は，それぞれ命令の左オペランドおよび右オペランドを示す。ここには前の命令の実行完了によってデータがセットされる。試作するマシンの基本データ語長（オペランドサイズ）は16ビットとした。

OPR1およびOPR2には図5.3(3)のようにオペランドタイプ領域がある。このタイプには例えば，制御コード，Boolean，Character（2文字まで），Short Integer（16ビットまで），Long Integer（32ビット），Real（32ビット），string（任意長の文字列），Pointer，Array等がある。これらのうち，オペランドに直接データを格納できるタイプ（Boolean，Character，Short Integer，制御コード，pointer）をShort Data Typeと呼び，それ以外をLong Data Typeと呼ぶ。Short Data Typeの場合は，Type Specifier（4ビット）の次にデータのビット長（4ビット）がセットされる。

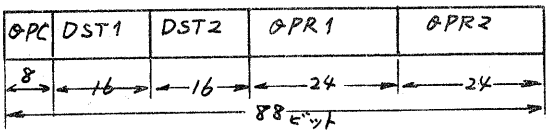
Long Data Typeの場合はType Specifierの次に20ビットのデータ空間のバイトアドレスがセットされる（データの実体はデータ空間内に格納される）。

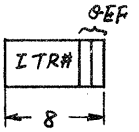
5.5 PU

PUはIMより送られた命令をデコードし，実行する。PUは複数個存在し，各々は独立に動作する。PUがBusyでないとき，その旨が全てのIMにBroad Castされる。IMは実行可能命令があればその命令をPUに転送する。PUは送られた命令を受取り，そのオペコードに応じて演算しその結果をR-Busを介してIMに転送する。通常の命令はdestinationが2つまで指定できるので，最大2個の結果が送られる。

命令がfunction callまたはreturn命令のときは，それぞれ，子functionを起動するために，または，親functionにresult list

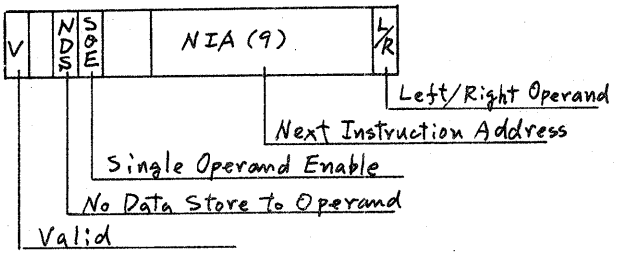
(1) 命令フォーマット及びEFTエントリ





EFTエントリ

(2) DST1, DST2のフォーマット



(3) OPR1, OPR2のフォーマット

(a) Short Data Type のとき

Type (4)	length (4)	data (16) (下位 length ビットだけ有効)

(b) Long Data Type のとき

Type (4)	Pointer to Data Space (20)

図5.4 命令フォーマット

を返すために，PUはLCPに割込みをかけて処理を依頼する。 call命令のときはargument listのpointerが，return命令のときはresult listのpointerがそれぞれLCPに渡される。

PUは，命令実行終了後，R-Busに結果を命令で指定されたDSTと共に出力する。DST1及びDST2の両方がvalid=1のときは2回結果が出力される。 R-Busに出力される情報のフォーマットは，図5.4の(2)及び(3)と同様であるが，DST部の上位2ビットに以下の信号が付与される。

- ICP (InsTruction Completed)：命令実行が完了したか否かを示す。DSTが2つのときは最後の結果にセットされる。
- R/W (Read/Write)：IMをReadするかWriteするかを指定する。

PUは，独立に動作可能なものが複数個存在するが，各PUは全てのマシン命令をインタプリトできるような汎用機能のものが望ましい。
しかし，PUを例えば，マイクロプログラム制御で実現するとすれば，それぞれのPUの制御メモリの容量が増加し，1つのLSI中にDFE全体を収容することが困難となろう。一方，各PUの機能を専用化すればPUの競合が発生する確率が高くなる。
従って，使用される頻度の高いプリミティブ命令は，複数のPU間で負荷分散し，特殊な機能は特定PUに専用化させるような構成を検討する必要があろう。

5.6 データキャッシュ

DC (Data Cashe) は，PUがLong Data Typeのデータを処理する場合のキャッシュとして使用される。 Long Data Typeの場合，データは仮想データ空間(図3.1参照)に格納されており，オペランドには，このデータ空間内のアドレスがセットされている。DCとは，いくつかのブロックに分割されており最近アクセスされたデータを含むブロック群を保持している。
オペランドがpointerであるとき，まず指定されたアドレスを含むブロックがDC内に存在するか否かが調べられる。 ブロックがない場合は，LCPにデータ要求の割込みが発生する。LCPは，必要なデータを含むデータブロックを外部インタフェースを介してDFEのDC内へ転送する。(このとき，必要があれば，DC内の他のブロックが仮想空間内に追出される。)
return命令を実行したとき，あるいは，

functionがDFEからswap outされたときは，DC内の変更ビットがオン（すなわちデータブロックの内容に変更があった）になっているブロックは全て仮想空間内に追い出され，DCの内容はリセットされる。

5.7 Bus

Busには，図5.1のようにI(Instruction)-BusとR(Result)-Busがある。I-Busのデータ巾は88ビット，R-Busのデータ巾は40ビットである。共通バス方式であるので，バスの転送速度は各PUやIMの速度に比して十分高速でなければならない。

5.8 実行可能命令のカウンタ

各DFEは，実行可能状態又は現在実行中である命令数を保持するためのカウンタEIC（Enabled Instruction Counter）を持っている。
実行中のfunctionにおいて，functionCall命令以外の命令が実行可能（又は実行中）の状態でないとき，そのfunctionは子functionの終了待ちの状態にある。このとき，待ち状態のfunctionはスワップアウト可能であり，他のfunction起動要求によりスワップアウトされることがある。
EICの値は，IECが命令の実行可能状態になったことを検出したときインクリメントされ，命令の実行が完了して，R-Bus上にICP（5.5参照）指定が現われたときにデクリメントされる。
DFEは，実行中の全命令がcall命令であるかどうかを検出するため，EICの他に実行中のfunction call命令のカウンタCIC（Call Instruction Counter）を持っている。CICの値は，PUがcall命令を実行したときインクリメントされ，子functionよりresult listが返されたときデクリメントされる。
EICの値とCICの値は常に比較されており，両者が一致したときLCPに割込み信号を送り，自functionがスワップアウト可能になった事が通知される。
このほか，DFEには，オン状態になっているDEFの数をカウントするためのEFC（Enable Flag Counter）を持っている。EFCはreturn命令を実行した際に，残留トークンがあるかどうかのチェックに用いられる。（ITR#がゼロで，しかもEFCがゼロでない場合はプログラムエラーである）。

5.9 外部インタフェース

DFEの外部インタフェース線は図2.2のD-Busへ接続される。DFEはこの外部インタフェースを介してLCPと通信する。この外部インタフェースを使用するのは，例えば，プログラムのロード／スワップアウト，データキャッシュへのロード／ストア，function call／return命令の実行，あるいは残留トークンのSave／recoverの場合である。
DFEは，最終的には1チップLSI化をねらっており，ロジック／ピン比を高くするため，データ巾はあまり大きくしない方が望ましい。このデータ巾は8ビット程度が適当であると思われる。

IV. 実験システム

D³Pの第1versionとして図6.1のような実験システムを試作し，DFE及びLCPの評価を行う。

HOSTより，初期データ及びデータフロープログラムが与えられたとき，LCPはまずDFEに親functionを割当てる。DFEで実行される親functionは必要に応じてfunction call命令を実行して，子functionを起動するようにLCPに依頼する。

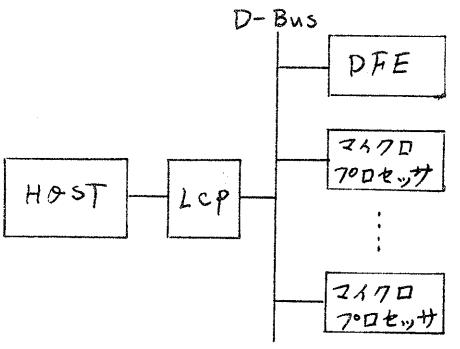


図6.1 実験システム

起動された子functionは，通常のマイクロプロセッサの1つに割当てられシーケンシャルに実行される。

このシステムでは，DFEに割当てられたfunctionは命令レベルで並列実行できるが，マイクロプロセッサに割当てられたfunctionはシーケンシャルに実行される。
プログラム構造を適当にすれば，DFE function の実行によって次々とマイクロプロセッサのfunctionが起動され，各マイクロプロセッサは非同期的に動作する。すなわちDFE（およびLCP）は一種のタスクスケジューラとして動作する。

Ⅶ. アプリケーション

D³Pの適用領域としては，数値処理，非数値処理（リスト処理）の両分野を考えている。
数値処理の場合，浮動小数点データのような表現法は必須のものであるが，プロトタイプシステムでは基本データ幅を16ビットとしたためこのようなデータへのアクセスはポインタ形式となる。ポインタ形式では，アクセス時のオーバヘッドが問題となるが，このオーバヘッドはデータキャッシュにより軽減できよう。

非数値処理としては，LISP等の関数型言語を用いた専用システムを考えている。専用システムとは，非同期プロセスから構成された一種のソフトウェアによるプロセッサの構成であり，広範な適用が可能であり今後検討を進めていく。

Ⅷ. おわりに

高並列処理が可能なD³Pシステムアーキテクチャを検討したが，これからLSI化への検討を含めてさらに詳細なハードウェア仕様を決定していく予定である。

現在のD³Pシステムには，なおいくつかの問題が残されている。

(1) CPヌはLCPのオーバヘッド
D³Pは集中管理システムであるので，functionの割当て時や解放時にCPヌはLCPに制御が渡される。従って起動されるfunctionの数が大きくなると，このオーバヘッドが無視できなるという問題が残される。

(2) Stream [8]
多数のデータセットを同一functionに対して apply するような場合には，Stream的手法は非常に有効であると思われる。
しかし，Streamのインプリメントにおいては連想記憶等の比較的高価なハードウェアサポートが必要となる。

(3) 非同期的 function call
LCPのオーバヘッドを軽減するために，function call は，そのスカラーギュメントが全てそろった時点で発生させ，functionからのreturnは全ての結果が得られてから実行するようにしている。この方式では，並列性の一部が失なわれる恐れがある。

(4) 構造体データ
array や tree 等の構造体データをfunction間で受渡しする場合，side effectの問題を避けるために，原則としてその実体をコピーする。すなわち，各functionはそれぞれ独立のデータ空間を持っており，参照されるデータは全てその空間内で閉じている。しかし，このコピー動作は構造体サイズが大きくなるとオーバヘッドが大きいので，例えば reference count [5] のような何らかの共有制御方式も検討する必要があろう。

(5) データフロー言語
データフロー言語として，当面はアセンブラ言語レベルのものを設定する。将来は高級言語の検討も必要であろう。

<参考文献>

[1] J.B. Dennis, "The Varieties of Data Flow Computers", The First International Conference on Distributed Computing Systems, Oct., 1979.

[2] J.B. Dennis, et al, "A Preliminary Architecture for a Basic Data-Flow Processor", Proceedings Second Annual Symposium on Computer Architecture, Jun., 1974.

[3] G.S. Miranker, "Implementation of Procedure on a Class of Data Flow Processors", Laboratory for Computer Science, MIT, July, 1975.

[4] J.B. Dennis, et al, "A Computer Architecture for Highly Parallel Signal Processing", Proceedings of the ACM 1974 National Conference,

[5] J. Rumbaugh, "A Data Flow Multiprocessor", IEEE Trans. on Computers, Vol. C-26, Feb., 1977.

[6] J.C. Syre, et al, "Pipelining, Parallelism and Asynchronism in the LAU System", 1977 International Conference on Parallel Processing, Aug., 1977.

[7] W.F. Coté, et al, "The Design of a Data Driven Processing Element", 1978 International Conference on Parallel Processing.

[8] Arvind, et al, "An Asynchronous Programming Language and Computing Machine", Department of Information and Computer Science, Univ. of California, Irvine, Dec., 1978.

[9] P.C. Treleaven, "Principal Components of a Data Flow Computer", Euromicro, 1978.

[10] A.L. Davis, "A Data FLow Evaluation System Based on The Concept of Recursive Locality", Proc. of 1979 NCC.

[11] I. Watson, et al "A Prottype Data Flow Computer with Token Labelling", Proc. of 1979 NCC.