電子装置設計技術 10－1
（1981. 10. 20）

パイプライン計算機設計支援システム（PSDS）

小栗 清　仲西秀基　田尻和夫　進藤 章　中村行宏
（日本電信電話公社　横須賀電気通信研究所）

1．はじめに

集積回路技術の進歩により、計算機がＬＳＩさらにはＶＬＳＩで作られる様になると、ゲート単位でのハードウェアコストの低下から、より複雑な論理回路の利用が促進されるために、また製造後の修正が困難であるために、いかに能率よく、また間違いのない論理設計を行うかが重要な課題になるものと思われる。

ところが、論理設計におけるＣＡＤとして種々のシステムが提案されているが、実用計算機の設計に適用された例はあまりない。[1],[2],[3]

この様な状況は、設計者の慣れによるところもあろうが、やはりこれまでに提案された言語の記述力、あるいは記述されたものの読み易さが劣っていたためであると思われる。特に装置内部が複数の制御部から構成され、それらが互いに連携をもって並列動作する様な対象の記述が困難であった。

ここでは当研究室での設計経験をもとに研究を進めている、パイプライン構造の計算機の記述に適した言語について述べる。

2．構造と動作

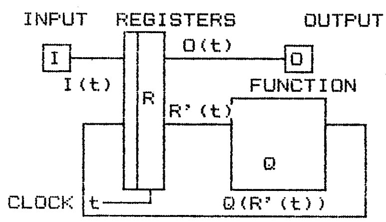

FIG-1. MODEL OF DIGITAL SYSTEM

まず、ゲートによる回路図によって全てのデジタルシステムが記述できる様に、全てのデジタルシステムが記述できる記述形式を考える。ただしシステムは、論理的には単相のクロックに同期して動作するシステムを対象とする。この場合、デジタルシステムのハードウェアの構造はＦＩＧ－１の様に表現することができる。クロック（ｔ）が通過して確定したレジスタ群の値Ｒ´（ｔ）が組み合せ回路群Ｑに入れられ、一定時間後にはＲ´（ｔ）の論理をとった結果であるＱ（Ｒ´（ｔ））が確定し、これは入力Ｉ（ｔ）と共にレジスタ群の全体Ｒに入力される。次のクロック（ｔ＋１）によって、これらはレジスタ群の全体Ｒの値となる。この動作の繰り返しがそのシステムの動作そのものである。以上の動作は

$$R(t+1) <= (\ Q(R'(t)),\ I(t)\ )$$
$$R(t) = (\ R'(t),\ O(t)\ ) \text{ --------------- } (1)$$

と表現できる。ここで、Ｏ（ｔ）は出力を表わす。第２式はシステムの全レジスタが、組み合せ回路群の入力となる部分と出力端子へ出力される部分とからなることを示している。

ハードウェアの設計とは、ハードウェアの状態を決定するいわば変数R（レジスタ群）と、その時間変化を決定するいわば関数Q（レジスタ間をつなぐ組み合せ論理）を決定することである。またハードウェアの動作は、変数Rの時間tに対する振るまいであると言える。式（１）は

$$Ri(t+1) <= Qi(\cdots, Rj(t), \cdots\cdots, Ik(t), \cdots)$$
$$i = 1, 2, \cdots, l$$
$$j = 1, 2, \cdots, m$$
$$k = 1, 2, \cdots, n \quad \text{－－－－－－－－－－－－－－－－－－（２）}$$

の様な式の集合である。ここでRiはレジスタ、Qiは組合せ論理を表わす関数、lは全レジスタ数、mはR´に含まれる全レジスタ数、nは入力の数である。普通は動作に大きく影響をあたえるレジスタ（制御レジスタ）Rcに着目して（２）を

$$Ri(t+1) <= pi(\cdots, Rj(t), Ik(t), \cdots) \cdot \underline{qi(\cdots, Rc(t), \cdots)} + Ri(t) \cdot qi(\cdots, Rc(t), \cdots)$$
$$i = 1, 2, \cdots, l$$
$$j, c = 1, 2, \cdots, m \quad \text{ただし} \quad j \neq c$$
$$k = 1, 2, \cdots, n \quad \text{－－－－－－－－－－－－－－－－－－（３）}$$

と考えることが多い。pi、qiはQiを分解したものである。この場合には

$$qi(\cdots, Rc(t), \cdots) : Ri(t+1) <= pi(\cdots, Rj(t), Ik(t), \cdots)$$
$$i = 1, 2, \cdots, l$$
$$j, c = 1, 2, \cdots, m \quad \text{ただし} \quad j \neq c$$
$$k = 1, 2, \cdots, n \quad \text{－－－－－－－－－－－－－－－－－－（４）}$$

と書くことができる。：は条件式と実行式を区切る記号であり、式（４）は条件式（左辺）が真のとき実行式（右辺）を実行するとの意味である。この表現法はChuによって提案されたＣＤＬ[4]と同様のものである。式（４）の形式（およびＣＤＬ）はハードウェアの構造をそのまま表しているため、いかなるデジタルシステムのハードウェアの構造も記述できる。

しかし、ハードウェアの構造そのものを表現する上記の記述形式では、システムの動作は、陽には表現できないため、全てのタイミングに対して、システムの目的とする動作を過不足なく行う共通のハードウェア構造の全体を記述しなければならない。その意味ではゲートレベルの回路図による記述と同様であるが、慣れや読み易さの点から、このままでは使い難いと思われる。これを解決するには、動作を表現できる記述形式が必要である。

３．パイプライン計算機と設計上の概念

計算機に関し、①　機能を持った各種資源とその制御構造を合せたもの（これを制御主体と呼ぶ）、②　ある目的を達成するために、必要な各種資源を使用したいという意志又は要求（これをハードウェア・タスク（略してタスク）と呼ぶ）、という概念を考えると、「計算機による全ての処理はタスクが、必要な制御主体を通過して、その管轄下にある各

種資源を動作させることにより実行される」と言うことができる。

一般に、計算機は、実行すべきタスク群に対して、必要な複数の制御主体を設けて構成することができる。

以上の考察から、計算機の動作・構造を表現するとは、制御主体の動作・構造を、タスクとの関係において記述することであると言える。既存の設計言語（例えば、ＤＤＬ[5]、ＦＤＬ[6]等）では、複数のタスクとそれを処理する複数の制御主体との関係を明確に記述することができない。これは、設計言語の仕様の決定に当って、ここで述べたような「ハードウェア・タスク」という概念を明確に意識していなかったことが原因と考えられる。以下、制御主体にはタスクとの関係においてどの様な構造が必要であるかを考察する。

ユニットに転送されたいくつかのタスクに共通な処理がある場合に、これを一つのタスクとして扱い、共通処理を必要とするタスクからこのタスクへの依頼ができる様にすると制御が簡単化できる。この依頼を受けるタスクを子タスクと呼ぶ。

パイプライン構造の制御主体とタスクとの関係は次の様である。パイプラインは、リソースをうまく配置して、どのタスクに対してもほぼ同様に何段階かの処理を施す様にし、順次タスクを投入して、同時に処理させる様にしたものである。パイプラインのリソースに対応する一段分をパイプと呼ぶ。パイプの周りには、タスクが、処理を依頼するために他のタスクの終了を待つ、処理を依頼した子タスクの終了を待つ、優先順序に従ってパイプの使用を待つ、パイプでの動作を監視する、追い越してはならない他のタスクの終了を待つなどのための構造が必要である。これをポートと呼ぶ。他のタスクに追い越されないよう監視するための構造をチェッカと呼ぶ。この他にパイプへのタスクの投入をコントロールするアービタや同じ優先順位のタスクの待ち行列を構成するキューが必要である。

以上のパイプラインシステムの構成要素が更に詳細にはどのような機能、構造を持つ必要があるのかは４章「ＰＳＤＬの概念」で述べる。

ここで述べた様にパイプラインシステムでは、リソースに対応して制御主体も分散させる構造とすることが多い。この様にデータと制御情報が共に存在し、順次流れていく様な構造は、遠くへ制御線を引きまわす必要がなく、今後のＶＬＳＩに向いているものと思われる。なお全体を一つのマイクロプログラムでコントロールする装置、すなわち制御主体とタスクが共に一個であるシステムは、以上に述べたシステムの縮退形として扱うことができる。これらの理由から、パイプライン構造を設計対象としてとりあげ、記述能力の目標とした。

４．ＰＳＤＬの概念

３章で述べた「タスクと制御主体」という概念により、パイプライン計算機の動作・構造を表現できることに着目し、このような対象の設計に適した記述言語について考察する。なお設計の進め方に関しては、実用性を考慮して次の様な階層を想定する。

①　対象とする論理装置は、タスクと制御主体により記述できるいくつかの部分（以下

ユニットと呼ぶ、例えば　　ストレージコントロール部）に分割して、設計する。ユニット間のインタフェース信号線を記述することによって、装置全体の構造を記述する。

②　ユニット内の設計に関しては、さらに内部をいくつかの部分（以下ステージと呼ぶ）に分割して、設計する。前出のポート、パイプ、キューなどはそれぞれステージである（ユニット、ステージの詳細は、以下で述べる）。

このような設計作業において、設計言語の望ましいイメージは、

①　各ステージの中を、順次、タスクが通過していく様子を記述することにより、各ステージを構成する各種資源とその動作を表現することができる、

②　全ステージについて、（１）のように記述し終ったときに、対象としたユニット全体が記述できている、

ということになる。このような言語として、ＰＳＤＬ（Pipeline　Structure　Description　Language）を考察した。以下その内容を示す。

４．１　ステージ

３章『パイプライン計算機と設計上の概念』で示したポート、キュー、アービタ、パイプ、チェッカの様な構造も結局は２章『構造と動作』で示した様なレジスタとレジスタ間の論理を決定する組み合せ論理、すなわちクロック毎の動作を決定する構造から成り立っている。ポート、キュー、アービタ、パイプ、チェッカ等は、この構造の内の個々の動作の特徴に着目して分類した部分である。ステージはこれらの一般名である。

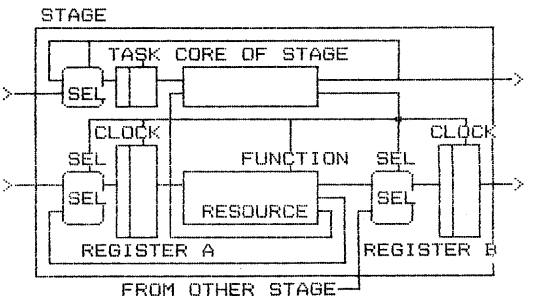

FIG-2. STAGE STRUCTURE

ステージの構造は一般にはＦＩＧ－２の様に考えることができる。ステージはタスクの処理対象となるレジスタ群、これに対して処理を行うリソース群および処理結果を格納するレジスタ群（ここまでをデータ構造と呼ぶ）、そしてレジスタ群、リソース群の間に存在するセレクタ（セレクタ自身はデータ構造に含まれるが、ＰＳＤＬでは自動発生されるので記述する必要はない）のセレクト端子、リソースの機能を決定するファンクション端子、レジスタのクロック端子（これらを制御ポイントと呼ぶ）に対し、このステージに滞在しているタスクの要求に応じた信号を発するステージのコア、および、タスクの滞在と状態を表わすタスクレジスタから成り立っている。キュー、ポート、アービタ、パイプ、チェッカのユニットに占る例をＦＩＧ－３に示す。各々の機能は以下の様である。

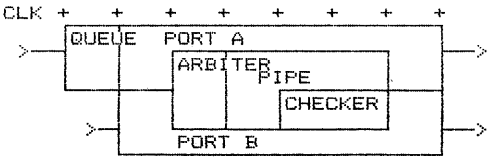

FIG-3. RELATION OF MAIN STAGES IN UNIT

①　キューは同種のタスクの待ち行列を構成する。

② ポートはタスクの種類毎に存在し、クロック毎に状態を変えながら、そのタスクの処理に関するいっさいのコントロールを行う。例えば、そのポートのタスクがパイプ内で問題を起した場合の処置も行う。

③ アービタはパイプの使用を待っているタスクを、優先順位に従ってパイプに入れる。

④ パイプは本来の目的とする処理を行い、大きなリソースに対応している。

⑤ チェッカは、実行上置いてきぼりにされたタスクが、論理的には、後のタスクに追い越されない様監視する。

４．２ ユニット

ステージによる動作の記述の範囲をユニットと呼ぶ。ユニットはいくつかのステージによって構成され、ユニット内には普通複数個のタスクが存在する。他のユニットとの結合はインタフェース信号線名で行うこととし、仕様は日本語で記述する。装置の記述が進んだ段階で各ユニット間のインタフェース信号線を取り除いて、合せて一つのユニットの記述とする事も可能とする。また、ＰＳＤＬではユニットを単位として、これを信号線名によって結合することによって階層構造を作る事ができる。

４．３ ＰＳＤＬの記述項目

ＰＳＤＬではユニットに関し

① ユニット名および管理情報

② インタフェース信号線

③ ステージの構造

④ レジスタ、メモリ

⑤ リソース

の各情報を記述する。以下に各項目の説明を行う。

①．ユニット名および管理情報

プロジェクト名、ユニット名、版数、年月日、作成者等を記述する。

②．インタフェース信号線

インタフェース信号線は＜ＯＵＴ＞、＜ＩＮ＞、＜ＣＯＭ＞に分けて書き並べる。その意味はステージの中で記述する。ただし特別な信号線として、＜ＲＥＳＥＴ＞、＜ＣＬＯＣＫ＞、＜タスク転送＞、＜ＳＣＡＮ ＡＤＤＲＥＳＳ＞、＜ＳＣＡＮＯＵＴ ＤＡＴＡ＞、＜ＳＣＡＮＩＮ ＤＡＴＡ＞、＜ＳＣＡＮＩＮ ＣＬＯＣＫ＞ がある。これらはその意味があらかじめ決められているものである。ＲＥＳＥＴは非同期リセットであり全てのレジスタをクリアする。ＣＬＯＣＫは論理的に単相のクロックであり、必要本数分だけ書き並べる。ＳＣＡＮ ＡＤＤＲＥＳＳはユニット内の全レジスタに設けたスキャンアウトパスに対し各レジスタを指定するアドレスである。ＳＣＡＮＯＵＴ ＤＡＴＡはスキャンアドレスで指定されたレジスタの内容を出力するデータラインである。ＳＣＡＮ ＩＮはスキャンアドレスで指定されたレジスタにスキャンインクロックにより書き込むデータである。ＳＣＡＮＩＮ ＣＬＯＣＫはスキャンインのためのクロックである。タスク転送

はこのユニットへの起動入力であり、転送先ステージ名とタスク名を記述する。

③．ステージの構造

ステージは、一般には、次の様に記述する。

＜stage　name＞ステージ名：種類［存在可能タスク名，……］

＜stage　execution　&　propagation＞

（状態名）

条件式：実行式；

……

a　種類はコンパイラでのチェックに使用される。

b　［存在可能タスク名］では、このステージに存在することのできる全タスクを宣言する。配列の場合は、タスク名［０－１５］の様に記述する。ステージの動作はタスクが存在している時にのみ実行される。

c　（状態名）はステージの状態毎に、その名前を記述する。状態である以上、他の状態とは排他的である。状態名の中には予約語として［タスク名］・（ＥＭＰＴＹ）を置く。これはそのステージにそのタスクが存在しない状態を表わし、他のステージから参照される。

d　条件式は、レジスタ名＝値、レジスタ名＝レジスタ名、レジスタ名（１ビットの場合）、［タスク名］（存在可能タスクが複数の場合に必要）、（状態名）（自ステージの状態名が条件式に含まれることはない）、信号線名、リソース名で構成される論理式である。

e　ステージ名・を修飾表現として、条件式では、レジスタ名、状態名、信号線名に付けることができる。

信号線名（端子を含む）、リソース名は小文字、その他は大文字で記述する。

条件式は次の様な階層表現が可能である。

```
ＲＥＧ１＝４          ：　実行式　；
ＴＲＵＥ              ：　実行式　；
１＊ＲＥＧ２          ：　実行式　；
  ２＊ＲＥＧ３＝１     ：　実行式　；
    ３＊ＲＥＧ４       ：　実行式　；
      ４ＲＥＧ５＝６   ：　実行式　；
      ４ＲＥＧ６＝３   ：　実行式　；
    ３＊ＥＬＳＥ       ：　実行式　；
  ２＊ＲＥＧ３＝２     ：　実行式　；
  ２＊ＥＬＳＥ         ：　実行式　；
１＊ＥＬＳＥ          ：　実行式　；
```

同じ数字で始まる条件式は同じレベルであり、大きな数字で始まる条件式は直前の小さな数字で始まる条件式に含まれる。＊の付いた条件式は同じレベルの中で排他的であるこ

とを示す。数字のない条件式は最上位のレベルとみなされる。

実行式は次の様な構造を持つ。

a　レジスタ名＜＝リソース名（リソース名（……（レジスタ名）…）

b　信号線名

c　＝＞［タスク名］

e　＝＞ステージ名・［タスク名］

f　＝＞＝＞ステージ名・［タスク名］

g　［end ］

a はレジスタ間の処理と転送を表わす。b は条件式が成立した時、この信号線名が真となることを表わす。c はステージの他の状態への遷移を表わす。d はステージ内のタスクの転送を表わす。e はタスクの他のステージへの転送を表わす。f は子タスクの発生を表わす。g はタスクの消失を表わし、これは他のステージで信号線名［end ］として参照できる。

＝＞で表わされるタスクの転送では、転送元のタスクは消失する。＝＞＝＞で表わされる子タスクの発生では転送元のタスクは消失しない。＜＝で表わされるレジスタの書き込みでは、転送元のレジスタの値は次に書き込まれるまでは保存される。

リソース名（リソース名…）はリソースの接続を表わす。また、リソースに出力が２カ所以上ある場合には、リソース名［何番目かを表す値］の様に、またレジスタ名が配列の場合には、レジスタ名［何番目かを表わす値］の様にする。

以上の記述の意味は、次の様である。すなわち、どの様な構造をも記述できた式（４）の集合の中から、「ステージの動作は、そのステージを必要としているタスクが、存在している時にのみ、ステージの状態、タスクに応じて行われる」、「タスクが遷移する」という構造を取り出し、その他から分離したものである。これによって、タスクの記述が陽に行える。

④．レジスタ、メモリ

レジスタは次の様に宣言する。

＜reg ＞レジスタ名（ビット幅）

＝レジスタ配列名［添字値］（ビット幅）

＝レジスタ部分名（ビット幅）！！レジスタ部分名（ビット幅）……

レジスタ部分名は上位から記述する。また、配列と部分名の組み合せも可能である。

メモリは次の様に宣言する。

＜mem ＞メモリ名（アドレス幅：ビット幅）

＝メモリ名［添字値］（アドレス幅：ビット幅）

＝メモリ部分名（アドレス幅：ビット幅）！！メモリ部分名……

第２式は２ポートメモリ等の記述に使用する。！！は連結子である。ステージの中で、メモリの読み出し、書き込みは、各々、次の様に記述する。

REG<=MEM[ADDRESS]

MEM[ADDRESS]<=REG

⑤．リソース

リソースには算術演算組み合せ回路と論理演算組み合せ回路があり、各々、次の様に宣言する。

<art >リソース名［０－添字の最大値］（第１入力ビット幅，第２入力ビット幅，…）

：第１出力ビット幅，第２出力ビット幅，……）

添字の値：ラベル名；ＯＵＴ番号＝ＩＮ番号　算術演算子　ＩＮ番号……

：出力最上位演算桁位置－出力最下位演算桁位置，

添字の値：ラベル名；ＯＵＴ番号＝ＩＮ番号　算術演算子　ＩＮ番号……

：出力最上位演算桁位置－出力最下位演算桁位置，……

```
<stage name>  BI_QUEUE :  QUEUE  [  BI,  BI[0-15]  ]
<stage execution & propagation>
(TASK_IN)
  [BI]                  : REG[INP]<=in INP<=inc(INP) =>(WAIT) [BI]=>[BI[INP]];
(WAIT)
  [BI]                  : REG[INP]<=in INP<=inc(INP) [BI]=>[BI[INP]];
  PORT_A･[EMPTY]        : [BI[OUTP]]=>PORT_A･[BI] PORT_D<=REG[OUTP]
                          OUTP<=dec(OUTP);
  [BI]･」(PORT_A･[EMPTY])
  ･inc(INP)=OUTP        : =>FULL;
  」[BI]･(PORT_A･[EMPTY])
  ･dec(OUTP)=INP        : =>(TASK_IN);
(FULL)
  PORT_A･[EMPTY]        : [BI[OUTP]]=>PORT_A･[BI] PORT_D<=REG[OUTP] =>(WAIT);
```

FIG-4. DESCRIPTION OF QUEUE

```
<stage name>  IF_PORT  :  PORT  [  IF  ]
<stage execution & propagation>
(PIPE_BUSY_WAIT)
  pipe_grant            :  =>=>ATB･[IF] =>(ATB) ATB_DATA<=IF_PORT_DATA;
(ATB)
  * atb_not_found       :  =>(TR_PORT_BUSY_WAIT);
  * ELSE;               :  =>(LM);
(LM)
  * lm_not_found        :  =>(INT_BUSY_WAIT);
  * check               :  =>(CHEKER_WAIT);
  * ELSE                :  =>[end] =>(PIPE_BUSY_WAIT);
(TR_PORT_BUSY_WAIT)
   TR_PORT･[EMPTY]      :  =>TR_PORT･[TR] TR_PORT_DATA<=IF_PORT_DATA
                           =>(TR_PORT_WAIT);
(TR_PORT_WAIT)
   TR_PORT･[end]        :  =>(PIPE_BUSY_WAIT);
(INT_BUSY_WAIT)
   INT･[EMPTY]          :  =>=>INT･[BF] INT_DATA<=IF_PORT_DATA
                           =>(MS_DATA_WAIT);
(MS_DATA_WAIT)
   INT･(MOVE_IN)        :  =>(PIPE_BUSY_WAIT);
(CHECKER_WAIT)
   CHECKER･[end]        :  =>(PIPE_BUSY_WAIT);
```

FIG-5. DESCRIPTION OF IF_PORT

＜log＞リソース名［０－添字の最大値］（第１入力ビット幅，第２入力ビット幅，…）
：第１出力ビット幅，第２出力ビット幅，……）
添字の値：ラベル名；ＯＵＴ番号＝ＩＮ番号　論理演算子　ＩＮ番号……
：出力最上位演算桁位置－出力最下位演算桁位置，
添字の値：ラベル名；ＯＵＴ番号＝ＩＮ番号　論理演算子　ＩＮ番号……
：出力最上位演算桁位置－出力最下位演算桁位置，……
演算内容の記述の右辺では、どの入力も最下位を合せて演算するものとする。

ＦＩＧ－４，ＦＩＧ－５にＰＳＤＬでキュー、ポートを記述した例を示す。

５．おわりに

以上、パイプライン構造の計算機を記述するための設計言語（ＰＳＤＬ）について、その考え方を述べた。ハードウェア・タスクの概念を導入することにより、従来の設計言語では、いずれかにかたよりがちであった計算機の動作／構造を同時にかつ明確に記述できることを可能にした。この特徴により、並列処理を行うデジタルシステム一般の記述に適用できるものと考えている。

今後開発を進めるコンパイラ、シミュレータ、ゲート展開プログラムでは、記述の特徴を生かして、各々次の様な機能を実現し、設計支援システム（ＰＳＤＳ）としての完成を図る予定である。

コンパイラ：シミュレーション用の内部表現への展開の他、ステージ間接続図、レジスタ、メモリ、リソース間接続図の出力、状態遷移条件の過不足のチェックなどを行う。

シミュレータ：入力端子データを入力とし、クロック毎のレジスタ、メモリ、リソースの指定位置の値を出力とする通常のシミュレーションの他、レジスタ（書き込みのみ）、メモリ、リソース、のステージ間での競合チェックやステージ間のタスクの同期チェックなどを行う。

ゲート展開プログラム：リソース名の各々に対し、部品（ゲート、機能ブロック）間の接続を対応させることにより、その等価性の確認、ユニット全体の部品間接続を示すＨＳＬ[7]表現への展開を行う。

謝辞

日頃御指導頂くデータ処理方式研究室橘本昭洋室長、ならびに室員各位に感謝する。

参考文献

（１）上原、川戸：大型コンピュータなどの論理設計に適用できるＣＡＤシステム，日経エレクトロニクス，1979.11.25，pp.104 -130,1979.

（２）Ｔ．Ｍ．Ｗｉｌｌｉａｍｓ and Ｌ．Ｃ．Ｗｉｄｄｏｅｓ：Ｓｃａｌｄ，Ｓｔｒｕｃｔｕｒｅｄ Ｃｏｍｐｕｔｅｒ Ａｉｄｅｄ Ｌｏｇｉｃ Ｄｅｓｉｇｎ，Ｐｒｏｃ.15ｔｈ ＤＡ Ｃｏｎｆ.，pp.271-277，1978.

(3) a. Vacca and N. R. Lincoln : Supercomputer outdoes itself by designing its successor (Computer-aided design systems and simulation helped the Cyber 205 to hit the ground running), Electronics, pp.106-110, Jun.1981.

(4) Y. Chu: Introducing CDL, IEEE Computer, Vol.7, No.12, pp.42-44, Dec.1972.

(5) J. R. Duley and D. L. Dietmeyer: A Digital System Design Language (DDL), IEEE TRANS. on Computers, Vol. C-17, No.9, pp.850-861, Sept.1968.

(6) 畑浦、渡辺、菊池、遠藤、和田、杉浦：FOREST言語（FDL）トランスレータ，昭56信学全大予稿 457, pp.2-222, 1981.

(7) 唐津・須藤：LSI階層仕様記述言語（HSL），電気学会システム・制御研資，SC -81-12,1981.