設 計 自 動 化 22－2
（1984. 7. 17）

階層的ハードウェア設計言語H2DLの言語仕様
－ 内部仕様記述とハードウェア・プロセス記述 －

西尾 誠一　宮田 操　山崎 勇
（東芝 総合研究所）

1．はじめに

近年のLSI技術の急速な進歩により、VLSIのように大規模なシステムが実現可能となってきた。しかしながら、一方ではこのようなシステムの大規模化に伴い、機能設計，論理設計のフェーズがLSI開発上のネックとなってきている。このような事態に有効に対処し設計効率を高めるためには、設計の上位のレベルから階層的設計手法、および階層的設計手法をサポートするCADシステムを用いることが不可欠と考えられる。

このような見地から、我々は新たに階層的ハードウェア設計言語H2DL（ Hierarchical Hardware Design Language ）と、これに基づくCADシステムの開発を進めている。［1，2，3］

H2DLは、内部仕様記述，ハードウェア・プロセス記述，接続記述，インタフェース記述の4種類の記述より成るが、ここでは内部仕様記述とハードウェア・プロセス記述の言語仕様について述べる。

2．内部仕様記述とハードウェア・プロセス記述の概要

内部仕様記述は、記述対象となるモジュールの内部構造を考え、レジスタ転送レベルで機能を記述するものである。大規模なハードウェアを設計する場合を考えると、一般に、その内部の構成要素をすべてレジスタ，メモリ等のレベルで直ちに決定することは困難である。このため、設計初期の段階では、まず機能的にまとまりをもった、いくつかのブロックに分割する。この時点で、各ブロックの有する機能は明確になるが、その入出力端子の種類と数，制御方法等はさらに設計が進んだ後でなくては決定されない。従来のレジスタ転送レベルの言語では、このような構成要素を含んだレベルから記述することは困難であったが、H2DLでは、ハードウェア・プロセスの概念を導入し、内部仕様記述からこれを引用できるようにしたことにより、このような設計初期のレベルからでも適用できるようにしている。［3］ ハードウェア・プロセス記述は、その機能のみに着目し、ハードウェアでの実現方法，入出力端子や制御信号等は考慮しないレベルのハードウェアの機能抽象化表現である。［2］

ハードウェアの内部構造を想定して、機能を記述する場合は、ハードウェアの並列性を表現するために、非手続き的な言語が適しているが、内部構造を意識しない抽象レベルで機能を記述する場合には、機能をアルゴリズム的に表現するため、一般のプログラミング言語のような手続き的な言語が適している。このため、H2DLにおいては、内部仕様記述では非手続き的な記述形式を、ハードウェア・プロセス記述では手続き的な記述形式をそれぞれ採用している。

3．内部仕様記述の言語仕様

内部仕様記述は、レジスタ転送レベルの機能をDDL［4］風に記述するものであるが、上記のように記述対象モジュールの構成要素として、ハードウェア・プロセスを使用できるようにしていることが、大きな特徴である。以下において、この内部仕様記述の言語仕様を簡単に紹介する。

内部仕様記述の全体の構成は、以下のとおりである。

```
<INTS>  モジュール名；
    宣言部
    機能記述部
<ENDINTS>；
```

宣言部では、外部端子，記憶要素，ハードウェア・プロセス等の構成要素の宣言を行ない、機能記述部では、宣言部で宣言された構成要素を用いて機能を記述する。

3－1．宣言部

宣言部での記述を以下で簡単に説明する。

3－1－1．外部端子の宣言

記述対象モジュールの外部端子を下記の形式で宣言する。

・形式

```
<IN>    入力端子名並び；
<OUT>   出力端子名並び；
<BID>   双方向端子名並び；
```

外部端子の宣言としては、この他に<NIN>，<NOU>，<NBI>があり、それぞれ負論理の入力端子，出力端子，双方向端子を宣言できる。ここで、負論理で宣言した場合には、high-level →“0”， low-level→“1”に、その他の場合には、high-level→“1”， low-level→“0”に、それぞれ対応する。

3－1－2．内部信号の宣言

記述対象モジュールの内部の信号を宣言する。

・形式

<TER>　内部信号名並び；

3－1－3．記憶要素の宣言

記述対象モジュールの構成要素として、レジスタ，ラッチ，メモリ等の記憶要素を宣言する。メモリの宣言では、書き込み可能なメモリは<RAM>で、読出し専用のメモリは<ROM>で、それぞれ宣言する。

・形式

<REG>　レジスタ名並び；
<LAT>　ラッチ名並び；
<RAM>　RAM名並び；
<ROM>　ROM名並び；

3－1－4．ハードウェア・プロセス引用の宣言

記述対象モジュールの構成要素として、ハードウェア・プロセスを引用することを宣言する。

・形式

```
<HPT>
    ハードウェア・プロセス定義名
        ：ハードウェア・プロセス識別名並び；
    …
<ENDHPT>；
```

・例

```
<HPT>
    ADDER：ADR4［4］<0：3>，
           ADR8［8］<0：7>；
    ALU　：ALUA［8］<0：7>，
           ALUB［8］<0：7>；
<ENDHPT>：
```

この例では、ADDERという名前でハードウェア・プロセス記述（後述）により定義されたハードウェア・プロセスをADR4（その出力は、ビット0～ビット3の4ビット），ADR8（出力は、8ビット）という名前で引用すること、およびALUという名前でハードウェア・プロセスとして定義されたものをALUA，ALUBという名前で引用することを宣言している。ハードウェア・プロセス記述においては、ビット幅等をパラメータとして記述するパラメータ付きの定義が可能になっている。このようなパラメータ付きで定義されたハードウェア・プロセスを内部仕様記述において引用する場合には、パラメータ値を指定する必要があり、この例での“ADR4［4］”のようにして指定する。

3－1－5．その他

・ユーザ定義名

宣言部でユーザが定義する端子名，レジスタ名，ハードウェア・プロセス識別名等の名前は、16文字以内の英数字および下線からなるものとする。

以上、内部仕様記述の宣言部の言語仕様を簡単に紹介してきたが、内部仕様記述では、その他の宣言として、

・クロックの宣言

記述対象モジュールの内部にクロック・ジェネレータを持つ場合に、クロック名とともにクロックの周期，デューティ等を定義する。

・名前付けの宣言

サブ・レジスタ，レジスタの連結等に対して名前付けを行なう。

等がある。

3－2．機能記述部

機能記述部では、以下の3つのブロックにより、記述対象モジュールの機能を記述する。

・スタティック・ブロック：

状態に依存せずに常に評価・実行される機能を記述する。

・ステート・ブロック：

状態遷移表現により、各状態で評価・実行される機能を記述する。

・割込みブロック：

割込み的な動作を記述する。

各ブロックは、機能記述部中でそれぞれ複数個記述してもかまわない。

以下において、まず基本事項を説明した後、各ブロックについて説明する。

3－2－1．基本事項

機能記述部の基本事項として、数値の表現形式，式，文等について説明する。

3－2－1－1．数値

内部仕様記述では、整数の他に2進表現，8進表現，10進表現，16進表現によりビット列の数値を表わすことができる。

・形式

①2進表現　　ビット幅B 2進数
②8進表現　　ビット幅O 8進数
③10進表現　　ビット幅D10進数
④16進表現　　ビット幅X16進数

ここで、2進数は0，1、8進数は0～7、10進数は0～9、16進数は0～9，A～Fである。

2進表現，8進表現，16進表現においては、さらに“U”（不定），“Z”（ハイ・インピーダンス），“？”（ドント・ケア）が表現できる。

・例

```
3OZ  ……  “ZZZ”
8X?3 ……  “????0011”
```

3－2－1－2．式

式には、単純式と条件式とがある。

①単純式

宣言部で宣言されたファシリティや数値は、単純式である。また、これ等ファシリティ，数値とオペレータ，関数で表現されるものも単純式である。さらに、演算順序の優先の指定に（）を用いることもできる。

表．3－1にオペレーター覧を、また、表．3－2に関数一覧を示す。

②条件式

条件式には、通常のＩＦ式，CASE式の他に、特徴的なものとしてPROVISO式がある。

◆ＩＦ式

・形式

```
ＩＦ 単純式 THEN 式1
          ELSE 式2 ENDＩＦ
```

◆CASE式

・形式

```
CASE 単純式 OF
  値並び：：式
  …
ENDCASE
```

◆PROVISO式

・形式

```
式2 PROVISO 単純式
    THEN 式1 ENDPROVISO
```

上記のＩＦ式と等価であるが、式2 を特に際立たせたい場合に有効である。ハードウェアの記述においては、例えば、「通常はレジスタAREGの内容を式の値とするが、フラグFLGが“1”の場合は、例外的にレジスタBREGの内容を式の値とする。」といった例が数多く存在するが、PROVISOを用いると、

```
AREG PROVISO FLG
          THEN BREG
```

のように、通常はAREGの内容を式の値とする、ということを強調した記述ができる。

3－2－1－3．文

文には、以下のようなものがある。

①結合文

・形式

外部端子または内部信号 ＝ 式；

外部端子，内部信号に対して、データを結合する。

②転送文

・形式

記憶要素 ＝ 式；

記憶要素に対して、データの転送を行なう。

③プロセス起動文

ハードウェア・プロセスの機能（プロセス）を起動する。

・形式

ハードウェア・プロセス識別名．プロセス名（引き数並び）；

・例

```
ALUA．PADD（ATER，BTER）；
```

レジスタ，ラッチ等は、システム既知の特殊なハードウェア・プロセスとして扱われ、表．3－3に示すようなプロセスが、あらかじめ用意されている。

・例

```
AREG．PCLR（）；
```

④状態遷移文

ステート・ブロック（後述）において、状態の遷移を行なう。

・形式

```
NEXT 状態名；
```

⑤条件文

条件文には、条件式と同様にＩＦ，PROVISO，CASEがある。

◆ＩＦ文

・形式

```
ＩＦ 単純式
   THEN 文の並び
   ELSE 文の並び
ENDＩＦ；
```

◆PROVISO文

・形式

```
文 PROVISO 単純式
      THEN 文の並び
   ENDPROVISO；
```

◆CASE文

・形式

```
CASE 単純式 OF
  値並び：：文
  …
ENDCASE；
```

⑥複合文

・形式

```
BEGIN 文の並び END；
```

⑦空文

・形式

NULL；

3－2－1－4．AT指定

内部仕様記述では、転送文の転送タイミングやプロセス起動文の起動タイミングの指定を柔軟に表現できるよう考慮しており、このために、AT指定と呼ぶ表現方法を導入している。以下で、このAT指定について説明を加える。

①転送文のAT指定

転送のタイミングを指定する転送文のAT指定の形式を下記に示す。

記憶要素＝式　AT　単純式；

このAT指定の意味は、左辺の記憶要素が、レジスタ，RAMかラッチかにより異なる。

◆レジスタ，RAMへの転送の場合のAT指定

AT指定により記述された単純式が“0”から“1”に変化するタイミングで転送が行なわれる。

・例

```
AREG=DATAIN　AT　CLK；
RAM1［MAR］=BREG　AT　～CLK；
```

この例の場合、レジスタAREGへの転送は、クロック信号CLKの立上りのタイミングで行なわれ、RAM1のMAR番地への転送は、クロック信号CLKの立下りのタイミングで行なわれる。

◆ラッチへの転送の場合のAT指定

AT指定により記述された単純式の値が“0”の間は、右辺の式の値をスルーし、単純式が“0”から“1”に変化するタイミングで、右辺の式の値をラッチする。

・例

```
ALAT=DATAIN　AT　CLK；
```

この例の場合は、クロック信号CLKが“0”の間は、ラッチALATは入力DATAINの値をスルーし、CLKが“0”から“1”に変化するタイミングでラッチする。

②プロセス起動文のAT指定

プロセス起動のタイミングを指定するAT指定の形式を下記に示す。

ハードウェア・プロセス識別名．プロセス名
（引き数並び）　AT　単純式；

このプロセス起動文のAT指定は、AT指定で記述された単純式が“0”の間はプロセスのCPART（後述）を評価・実行し、“0”ら“1”に変化するタイミングでプロセスのSPART（後述）を実行することを示す。

・例

```
AREG．PINC（）　AT　CLK；
```

また、条件文や複合文に対しても、AT指定を記述することができ、この場合は、条件文，複合文内に書かれたすべての文に対して、そのAT指定が有効となる。また、AT指定が多重に行なわれた場合は、近い方の指定が有効となる。

3－2－2．スタティック・ブロック

内部仕様記述では、常に評価・実行すべき機能をスタティック・ブロックとしてまとめて記述できるようになっており、下記の形式で記述する。

・形式

```
<STC>
    文の並び
<ENDSTC>；
```

ここで、文の並びに書かれた各文は、並列に評価・実行される。また、AT指定を省略した場合は、AT　0とみなされる。

3－2－3．ステート・ブロック

通常の状態遷移による機能の記述を行なう部分が、ステート・ブロックであり、下記の形式で記述する。

・形式

```
<STT>　ステート・ブロック名：クロック指定；
    状態名：文の並び
    …
<ENDSTT>；
```

ここで、内部仕様記述中にステート・ブロックを1つしか記述しない場合は、ステートブロック名：　を省略できる。

ステート・ブロック中では，常にただ1つの状態をとり、クロック指定により指定された単純式が“0”から“1”に変化するタイミングで状態が遷移する。状態名：の後に書かれた各文は、その状態の間並列に評価・実行される。ここで、AT指定を省略した場合は、その状態の間“0”で、状態の後縁で“0”から“1”に変化するAT指定があるとみなされる。

また、ステート・ブロックでは、多相クロックを用いた場合に対しても、柔軟に対処できるよう考慮されている。

・例

```
<STT>　STT1：CLK1／CLK2；
  状態名：文の並び1 ／文の並び2
  ……
<ENDSTT>；
```

この場合、1相目（CLK1）で実行される機能は文の並び1 で、2相目（CLK2）で実行される機能は文の並び2 で、それぞれ表現される。

3－2－4．割込みブロック

従来のレジスタ転送レベルの言語では、割込み的な機能を記述することは、困難であった。例えば、「どの状態にいようとも割込み信号ＩＮＴ１が“１”になったら、強制的に次状態を割込み処理の状態ＩＮＴ１ＳＴとする。」といった機能を記述しようとすると、すべての状態に、この動作を示す状態遷移文を書く必要があり、非常に繁雑な記述となっていた。内部仕様記述では、下記の形式の割込みブロックを導入したことにより、このような問題を解決している。

・形式

```
<EXC>  割込み条件；
   NEXT  ステート・ブロック名：状態名；
   …
<ENDEXC>；
```

ここで、ステート・ブロックが１つのみの場合は、ステート・ブロック名を省略できる。

この割込みブロックで、上記の例を記述すると、

```
<EXC>  INT1；
   NEXT  INT1ST；
<ENDEXC>；
```

のように簡単に記述できる。

また、通常の割込み動作では、このな状態遷移のタイミングで割込み動作に入る場合のみならず、割込み信号が“１”になった時点で直ちに割込み動作に入る場合もある。内部仕様記述では、この種の割込みの機能も記述できるようになっており、キーワード“ＮＥＸＴ”の代りに“ＤＩＲＥＣＴ”を用いて表現する。

4．ハードウェア・プロセス記述の言語仕様

ハードウェア・プロセス記述は、ハードウェアの機能のみに着目し、入出力端子，制御方法等は考慮しないレベルのハードウェアの一抽象化表現である。ハードウェア・プロセスにおいては、①制御信号，制御方法の抽象化、②入出力端子の抽象化、③ハードウェアによる実現方法の抽象化を行なうが、これ等の抽象化を以下の方法により実現した。

①個々の機能をプロセスとして記述し、ハードウェア・プロセスを引用している内部仕様記述から直接これを呼ぶことを可能とした。

②入力データは、引き数の形で内部仕様記述の側から与えられ、出力データは、ハードウェア・プロセス識別名で内部仕様記述から参照できるようにした。

③プロセスの記述で必要となる記憶要素は変数とし、機能の記述は、通常のプログラミング言語のように手続き的に行なえるようにした。

以下において、この言語仕様を簡単に紹介する。

ハードウェア・プロセス記述の全体の構成は下記のとおりである。

```
<HP>  ハードウェア・プロセス定義名［仮パラメータ並び］<出力レンジ>；
      グローバル変数の宣言
      プロセス記述部
<ENDHP>；
```

また、プロセス記述部は、１つ以上のプロセスの記述からなるが、各プロセス記述の構成は下記のとおりである。

```
<PROC>  プロセス名（仮引き数並び）；
   ローカル変数の宣言
   <CPART>
      文の並び
   <ENDCPART>；
   <SPART>
      文の並び
   <ENDSPART>；
<ENDPROC>；
```

ハードウェア・プロセスの個々の機能は、プロセスとして記述される。プロセスの記述では、組合せ回路的な機能をＣＰＡＲＴ（Combinational logic PART）で、順序回路的な機能をＳＰＡＲＴ（Sequential logic PART）で、それぞれ記述する。ここで、各パートで書かれた文は、手続き的に実行される。ＣＰＡＲＴは、組合せ回路的な機能を記述するものであるため、評価期間（ＡＴ指定が“０”の間）中に引き数（入力データ）の値に変化が生じた場合には、再実行を行なう。

グローバル変数は、各プロセスから広域的にアクセスできる変数であり、その値はプロセス実行後も静的に保持される。これに対し、ローカル変数は、他のプロセスからは直接アクセスできない局所的な変数であり、プロセスが起動されているときのみ値が存在する動的な変数である。ハードウェア・プロセス定義名は、グローバル変数の一種であり、プロセス実行の結果をこれにアサインし、ハードウェア・プロセスの出力とする。

また、ハードウェア・プロセス記述では、出力のビット幅等を可変としたパラメータ付きの定義が可能であり、パラメータはハードウェア・プロセス記述全体にわたり、記号定数として扱うことができる。

4－1．グローバル変数の宣言

ハードウェア・プロセスのようなハードウェアの実現方法を意識しないレベルの記述においては、ビット毎の論理演算等のビット列を意識した記述とともに、制御変数や四則演算等では、ビット列を意識しないレベルの記

述も行なえることが望まれる。このため、ハードウェア・プロセス記述では、信号型と整数型の2種のデータ型を用いることができるようにしている。グローバル変数の宣言では、この2つの型に対応して、<SIG>, <INT>があり、下記の形式で宣言する。

・形式

<SIG> 信号型変数名並び；
<INT> 整数型変数名並び；

ここで、多次元配列も宣言できる。（配列の添字は常に0から始まるもとする。）

・例

```
<SIG>  A<0:7>, B[3]<1:8>,
       C[4, 5];
<INT>  X, Y[2], Z[2, 3];
```

さらに、必要ならば、初期値の指定も可能となっており、下記のように記述できる。

```
<INT>  X=0, Y[2]=[1, 2],
Z[2, 3]=[[1, 2, 3] [5, 6, 7]];
```

4－2．プロセス記述

プロセス記述の全体の構成やCPART，SPARTの意味等については、前述した。ここでは、その中で記述するローカル変数の宣言形式や式，文について簡単に触れる。

◆ローカル変数の宣言

ローカル変数の宣言の形式は、上記のグローバル変数と同様である。

◆式

信号型のデータに対しては、内部仕様記述と同様のオペレータ・関数の他に、信号型から整数型への変換を行なう型変換関数（STI，STI2S，STI1S，STIABS）が使用でき、整数型のデータに対しては、表．4－1，表．4－2，で示されるオペレータ，関数が使用できる。また、内部仕様記述と同様に条件式も記述できる。

◆文

文には、代入文，IF文，PROVISO文，CASE文，FOR文，WHILE文，REPEAT文，GOTO文，複文，空文等があり、豊富な制御構造が用意されている。

5．記述例

図．5－1に、簡単な計算機SIMPLEの内部仕様記述例を示す。また、図．5－2にはSIMPLEで引用している演算ユニットASUのハードウェア・プロセス記述例を示す。

6．おわりに

以上、H2DLの内部仕様記述，ハードウェア・プロセス記述の言語仕様について説明した。

内部仕様記述の特徴としては、

・構成要素としてハードウェア・プロセスを用いることができ、設計の初期の段階から用いることができる。
・AT指定により、レジスタ転送のタイミング等を柔軟に指定することができる。
・多相クロックの記述も容易に記述できるよう考慮されている。
・割込みブロックにより、割込み的な機能の記述も容易にできる。

等があげられる。

また、ハードウェア・プロセス記述の特徴としては、

・ハードウェアによる実現方法，入出力端子，制御方法を抽象化した記述が可能である。
・手続的に機能を記述できる。
・パラメータにより、出力幅等を可変とする記述ができる。

等があげられる。

現在は、種々の例題記述を通して記述能力のチェック等を行なっており、16ビットのマイクロプロセサ（PULCE）の内部仕様記述の場合、約1500行で記述できた。

今後の課題としては、ハードウェア・プロセスの記述能力の拡張等があげられる。また、ユーザ・インタフェースの向上の見地からは、構造化エディタ，図形表現の言語等の開発が重要となってくるものと考えている。

◆参考文献

[1] M. Miyata，S. Nishio，I. Yamasaki：“A Hierarchical Hardware Description Language”，IEC EDA84，pp.189 ～ 193，1984.

[2] 宮田，西尾，増渕他：“階層的ハードウェア記述言語H2DLの機能抽象化について”，情報処理学会 第28回全国大会，2P－5，1984.

[3] 宮田，西尾，山崎：“階層的ハードウェア設計言語H2DLの思想”，情報処理学会 設計自動化研究会資料22－1，1984.

[4] J. R. Duley，D. L. Dietmeyer：“A Digital System Design Language (DDL)”，IEEE Trns. on Comp.，C－17，No.9，pp.850 ～ 861，1968.

表．３－１　内部仕様記述におけるオペレーター覧

優先順位	記号	意　味
1	！！	ビット列の連結
2	～	否定
3	゛	選択
4	／	縮小
5	＝＝	等しい
5	～＝	等しくない
7	&	論理積
8	～&	NAND
9	～！	NOR
10	～@	一致
11	@	排他的論理和
12	！	論理和

表．３－２　内部仕様記述における関数一覧

種　類	関　数	意　　味
シフト	SHL（X，n，S）	Xを左に nビットシフトし、Sを fill-inする
	SHR（X，n，S）	Xを右に nビットシフトし、Sを fill-inする
	SLA（X，n）	Xを左に nビット算術シフトする
	SRA（X，n）	Xを右に nビット算術シフトする
	SLO（X，n，S）	SHL（X，n，S）の最上位に最後にシフト・アウトされたビットを付加し、結果とする
	SRO（X，n，S）	SHR（X，n，S）の最上位に最後にシフト・アウトされたビットを付加し、結果とする
	SLAO（X，n）	SLA（X，n）の最上位に最後にシフト・アウトされたビットを付加し、結果とする
	SRAO（X，n）	SRA（X，n）の最上位に最後にシフト・アウトされたビットを付加し、結果とする
	CIL（X，n）	Xを左に nビットサーキュレートする
	CIR（X，n）	Xを右に nビットサーキュレートする
比較	GT（X，Y）	X＞Yの時に１、その他の時は０とする
	GE（X，Y）	X≧Yの時に１、その他の時は０とする
	LT（X，Y）	X＜Yの時に１、その他の時は０とする
	LE（X，Y）	X≦Yの時に１、その他の時は０とする
コピー	CPY（X，n）	Xを n個コピーする
加減算	INC（X）	Xを＋１する
	DEC（X）	Xを－１する
	ADD（X，Y，C）	X＋Y＋C（キャリー）の加算を行なう
	SUB（X，Y，B）	X－Y－B（ボロー）の減算を行なう
	ADO（X，Y，C）	X＋Y＋C（キャリー）の加算を行ない、最上位ビットにキャリー・アウトを付加する
	SBI（X，Y，B）	X－Y－B（ボロー）を減算し、最上位ビットにボローを付加する
ディレイ	DEL（X，n）	Xに n時刻のディレイを付加する

ここで、X，Yは単純式、S，C，Bは１ビットの単純式、n は整定数　とする。

表．3－3　レジスタ，ラッチのプロセス

種　類	プロセス
シフト	PSHL (S)
	PSHR (S)
算術シフト	PSLA ()
	PSRA ()
サーキュレート	PCIL ()
	PCIR ()
非同期クリア	PCLR ()
非同期プリセット	PPRE ()
非同期ロード	PLD (X)
インクリメント	PINC ()
デクリメント	PDEC ()

ここで、Sはシフトイン・ビット、
Xはロード・データである。

表．4－1　ハードウェア・プロセス記述における整数型データのオペレーター覧

優先順位	記号	意　味
1	+	（単項演算子）
1	−	（単項演算子）
3	**	べき乗
4	*	乗算
4	／	除算
6	+	加算
6	−	減算
8	>>	より大
8	<<	より小
8	>=	以上
8	<=	以下
12	==	等しい
12	～=	等しくない

表．4－2　ハードウェア・プロセス記述における整数型データの関数一覧

種　類	関　数	意　味
対数	LOG (n)	底を2とした対数をとる
絶対値	ABS (n)	n の絶対値をとる
型変換	ITS (n , m)	n をビット幅m の符号無し2進数に変換する
	ITS2S (n , m)	n をビット幅m の2の補数表現の2進数に変換する
	ITS1S (n , m)	n をビット幅m の1の補数表現の2進数に変換する
	ITSABS (n , m)	n をビット幅m の絶対値表現の2進数に変換する

ここで、n は整数式、m は整定数　とする。

```
<INTS>  SIMPLE;                                    "簡単な計算機SIMPLEの内部仕様記述"

  <IN>  CLK, START, OC, SCLR;                      "外部端子の宣言"
  <OUT> OUTDAT<0:15>;

  <REG> ACC<0:15>, IR<0:15>,                       "記憶要素の宣言"
        MAR<0:9>, LOC<0:9>;
  <RAM> M[0:1023]<0:15>;

  <HPT>                                            "ハードウェア・プロセスの宣言"
        ASU : ASU1[16]<0:15>;
  <ENDHPT>;

  <EXC> SCLR;                                      "割込みブロックの記述"
        NEXT INIT;
  <ENDEXC>;

  <STT> CLK;                                       "ステート・ブロックの記述"

        INIT:   LOC.PCLR();
                IF START
                  THEN  NEXT ADR;
                  ELSE  NEXT INIT;
                ENDIF;

        ADR:    MAR = LOC;        LOC.PINC();
                NEXT IFH;

        IFH:    IR = M[MAR];      NEXT DEC;

        DEC:    CASE IR<0:5> OF
                   1::  NEXT AD;
                   2::  NEXT SB;
                   4::  NEXT LD;
                   8::  NEXT STR;
                  16::  NEXT BR;
                  17::  NEXT BM;
                  32::  NEXT INIT;
                ENDCASE;
                MAR = IR<6:15>;

        AD:     ASU1.PAD(ACC,M[MAR]);   ACC = ASU1;      NEXT ADR;

        SB:     ASU1.PSB(ACC,M[MAR]);   ACC = ASU1;      NEXT ADR;

        LD:     ASU1.PTH(M[MAR]);       ACC = ASU1;      NEXT ADR;

        STR:    M[MAR] = ACC;                            NEXT ADR;

        BR:     LOC = IR<6:15>;                          NEXT ADR;

        BM:     IF ACC<0> THEN  LOC = IR<6:15>;  ENDIF;
                NEXT ADR;

  <ENDSTT>;

  <STC>                                            "スタティック・ブロックの記述"
        IF OC
          THEN  OUTDAT = ACC;
          ELSE  OUTDAT = 0;
        ENDIF;
  <ENDSTC>;

<ENDINTS>;
```

図. 5－1　簡単な計算機ＳＩＭＰＬＥの内部仕様記述

```
1       <HP>     ASU[N]<0:N-1>;              "演算ユニットASUのハードウェア・プロセス記述"
2
3         <PROC>    PAD(X,Y);                "加算プロセス"
4               <SIG>    X<0:N-1>, Y<0:N-1>;
5               <CPART>
6                        ASU = ADD(X,Y,0);
7               <ENDCPART>;
8         <ENDPROC>;
9
10        <PROC>     PSB(X,Y);               "減算プロセス"
11              <SIG>    X<0:N-1>, Y<0:N-1>;
12              <CPART>
13                       ASU = SUB(X,Y,0);
14              <ENDCPART>;
15        <ENDPROC>;
16
17        <PROC>     PTH(X);                 "データ・スルー・プロセス"
18              <SIG>    X<0:N-1>;
19              <CPART>
20                       ASU = X;
21              <ENDCPART>;
22        <ENDPROC>;
23
24      <ENDHP>;
```

図. 5－2　演算ユニットASUのハードウェア・プロセス記述