設計自動化 55－1
(1990.12.12)

ハードウエア記述言語A^{2}DL

高橋隆一　村岡泰釈*　江尻雅晴*　林 孝雄*

日本電気（株）
C&Cシステム研究所

＊ 日本電気（株）
複合交換開発本部

あらまし　ハードウエア記述言語A^{2}DLは、ディジタルシステムをオートマトンの階層でモデル化するハードウエア記述言語である。記法は、CDLとDDLを融合したものになっており、特に、マイクロアーキテクチャの設計に適している。要求駆動と名付けたシミュレーション手法を前提に、静的な箇条書きをすることに第1の特徴があり、宣言部で、異なるモジュールの変数に直接アクセスすることを宣言して、バスを明示することなく厳密な階層構造が表現できる点に第2の特徴がある。状態単位に箇条書きされたコントロールフローを破壊しないハイレベルシンセシザの実現にも成功しており、言語仕様と処理系A^{2}DL－DAの基本機能／性能は実証済みである。

An Automaton Description Language (A^{2}DL)

Ryuichi Takahashi,　Yasutoki Muraoka*,　Masaharu Ejiri*,　Takao Hayashi*

C&C Systems Res. Labs.
NEC Corp.

4-1-1 Miyazaki, Miyamae ku
Kawasaki City 213 Japan

＊) Integrated Switching Development Div.
NEC Corp.

1131 Hinode, Abiko City
270-11 Japan

Abstruct　Hardware Description Language A^{2}DL models digital systems as hierarchical clusters of automatons, which are finite state machines with datapaths. In this HDL, CDL-like notations and DDL-like notations are fused to form static and precise expressions of microarchitectures of given digital systems. These expressions have been proved to be suitable for high speed simulation technique known as "demand driven". We have already succeeded to develop a high level synthesizer as well, which is aimed at realistic allocation of datapaths and controls without destroying the control flows specified by the automatons. Basic functions and performance of the DA system called A^{2}DL-DA has been already proved during real 5 LSIs design.

1．はじめに

半導体技術の進歩と市場ニーズの高度化に伴い、従来、人手で行われていた方式設計や機能設計を自動化する試みが急速に活発化している。図面を用いた設計では、設計上流工程の自動化には限界があるが、これに代わる手段として、ハードウエア記述言語を用いることができる。

A²DLは高レベル自動設計システムA²DL－DAの核として設計したハードウエア記述言語であり、特に、マイクロアーキテクチャの表現に優れている。ここに「マイクロアーキテクチャ」とはALUやシフタ、キューなどの機能ブロックからなるシステム構成を意味する。キャッシュの更新戦略を含むメモリアーキテクチャやバスアーキテクチャもこのレベルで問題にされる。その静的な記述は効率の良いシミュレータ実現を可能にし(6)、コントルールフローが厳密に表現できるなどの特徴と相まって、実用的なシンセサイザの実現をも可能にした。

本稿ではハードウエア記述言語A²DLの概要説明に続いて、全容をマニュアルの形式で紹介する。記述例は他の文献(3)、(6)を参照されたい。

2．A²DLの概要

ハードウエア記述言語A²DL（An Automaton Description Language）は、ディジタルシステムをオートマトンの階層でモデル化するハードウエア記述言語である。ここに「オートマトン」は、制御信号を発生する決定性有限状態機械から見えるデータパス系の動作を表現するモデルである。A²DLによって記述されたオートマトンはすべて、一斉に、並列に動作する。

記法は、古典的なハードウエア記述言語CDL(1)とDDL(2)を融合したものになっており、65個の予約語と72個の非終端記号を用いたLALR1で定義されている。

2－1．静的な箇条書き

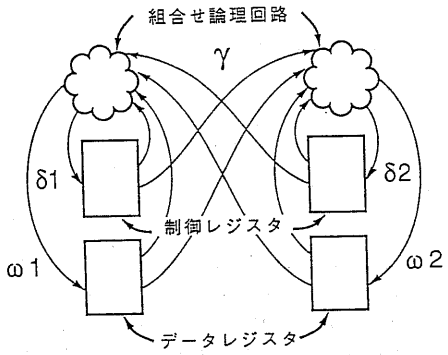
A²DLには、if then else、case、for、repeatなどの記法は存在しない。コントロールフローすなわち手続き的な動作は、オートマトンの状態単位に、そこで行われる動作を列挙することで表現される。論理設計もブール式の箇条書きで表現される。A²DLにイベントを意識した記法は存在しない。「いくつかの文が逐次的に動いて、最終的に所要の結果を与えるような書き方をしていない」という意味で、この箇条書きは「静的」である。

このような箇条書きが、同期式の設計に関する限り、記述能力の妨げにならないことは「オートマトンでモデル化し、制御レジスタ、データレジスタの内容を更新する関数を評価するだけで、システムの動きは完全に追跡できる」ことで保証されている（図1）。

A²DLのシミュレータは、このような箇条書きを前提に、必要最小限のブール式のみを、要求駆動で評価することで実現されている(6)。すべての関数は、評価要求が発生してはじめて評価され、入力信号の変化は無視される。要求駆動方式は、これと等価な論理シミュレーションより一桁高速である。

静的な箇条書きであるという特徴は、効率のよいシミュレータの実現だけでなく、実用的な合成をも可能にした。すなわち、各状態にそこで評価される、基本的な動作が明示されることでコントロールアロケーションをきわめて自然に行うことができる。また、論理設計を表現する箇条書きにおいて、適当なブール式に対し、その値が1になるような入力は禁止されている（don't careである）ことを、陽に指定できるため、これを考慮した論理簡単化を行うこともできる。

静的な箇条書きは、設計のレビューやクロスチェックをも容易にし、将来的には、設計無矛盾性の自動検証系の実現も保証している。



δ1、δ2：状態遷移に係わる関数
ω1、ω2：データ処理に係わる関数
γ：通信に係わる関数（δ、ωの一部分）

図1．オートマトンを構成する3つの関数

２－２．階層構造の表現

Ａ²ＤＬには、グローバルな変数という概念が存在せず、レジスタやメモリ、入出力端子は、必ずそれが存在するロケーションでローカルに宣言しなければならない。モジュール間の通信は、入出力端子間の結線（バス）を陽に表現するか、スコープルールを破って他のモジュールに直接アクセスするかの、両者が混在する場合を含む、２通りで表現される。この直接アクセスは、各モジュールの入出力宣言部で指定される。モジュール間の結線ですべての通信を表現できることは明きらかだが、直接アクセスの記法のみを用いてもすべての通信を表現できる。図１は、このことも説明している。

ローカルな宣言しかできないという制約は、厳密な階層構造の表現しか許さないための工夫である。結線が陽に表現されていない場合は、自動バス・アロケーションを前提としていると解釈される。特に直接アクセスの記法のみを用いている場合は、すべてのバスが自動的にアロケートされる。結線が明示されていれば、それらが優先される。

２－３．適用対象

Ａ²ＤＬで表現される仕様は、完全に決定的な動作をする。これは同期式の設計をはじめから前提としているためである。非同期的な信号が系に存在してもシステム本体が同期式なら、信号に同期してしか観測されないため、記述能力を制限することはない。

同期式の設計を前提とし、オートマトンをモデルとしていることは、クロック系（clocking scheme）をきわめて自然に表現することを可能にしており、マイクロアーキテクチャの表現能力を特に優れたものにしている。ある状態に、「Ａ：＝Ｂ，Ｂ：＝Ａ」と書けば、指定されたクロックのエッジで、内容は確実に入れ替わる。コントロールフローを厳密に表現していても、それを組み替える戦略を適用することも可能である。実際、オートマトンで表現された仕様からも、データ依存関係を抽出することで、スケジュリングなどの戦略も実施できる。２つのプッシュダンメモリをもつ、決定性有限状態機械はチューリングマシンと等価であることから、Ａ²ＤＬが任意のアルゴリズムを表現できることも保証されている。

３．語彙

３－１．予約語、識別子、即値数、コメント

予約語は６５個あり、うち３９個はオペレータである。オペレータには「＊」（AND）、「＋」（OR）、「@」（EXOR）、「’」（否定）という４個の特殊記号が含まれるが、他はすべて英小文字と数字のみからなる。以下にこれらを列挙する。オペレータに添えた矢印は、優先順位が強くなる向きを表している。「eq」（等しい）が最も弱く、「rev」（左右反転）が最も強い。

非オペレータ：

clear	inout	module	p(ositive)	virtual
clock	input	msb	register	wait
constant	logic	n(egative)	return	
duplicate	lsb	net	sector	
escape	main	operator	state	
exit	memory	output	use	

オペレータ：

```
(lowest)
    ¦ eq        ¦ coin      ¦ div       ¦ sizel
    ¦ ne        ¦ if        ¦ cry       ¦ copy
    ¦ gt        ¦ ror       ¦ mod       ¦ ""
    ¦ ge        ¦ rand      ¦ shl0      ¦ tc
    ¦ lt        ¦ rxor      ¦ shl1      v rev
    ¦ le        ¦ ld0       ¦ shr0   (highest)
    ¦ comb      ¦ ld1       ¦ shr1
    ¦ "+"       ¦ add       ¦ rotl         set ( )
    ¦ "*"       ¦ sub       ¦ rotr         reset ( )
    v "@"       v mul       v size0        -> dc
(continue)   (continue)  (continue)        := noc
```

英大文字ではじまり、英大文字と数字そしてアンダバーからなる文字列を、識別子（ＩＤ）に用いることができる。識別子は、後述する、モジュールやファシリティ（レジスタ、組合せ論理回路など）の名前に用いる。識別子はモジュール内で一意的であればよい。

即値数（Ｎ）は自然数に限られ、２進、４進、８進、１０進、１６進の５通りがある。１０進数はそのまま表現し、他はそれぞれ、

ｂ（「０、１の列」）・・・２進

ｔ（「０、１、２、３の列」）・・・４進

ｏ（「０～７の列」）・・・８進

ｈ（「０～９、Ａ～Ｆの列」）・・・１６進

という記法を用いる。ビット長は、１０進数は２進数での最小桁数、他は順に１の倍数、２の倍数、３の倍数、４の倍数と見なされる。

改行、空白は意味をもたない。空白を書くことのできる任意の箇所に、引用符「”」で前後をくくって自由にコメントを記入できる。

3－2．オペレータと式

A²DLにおいて「式」はｓｅｔ、ｒｅｓｅｔ、ｄｃ、ｎｏｃを除く、３５個のオペレータと、後述する、ファシリティの参照とから帰納的に次のように定義される（１）即値数とファシリティの参照は「式」である。（２）「式」に対しオペレータを作用させたもの及びそれを括弧「（）」でかんこんだものも「式」である。（３）上記（１）、（２）で作られるものだけが式である。

A²DLでは、この「式」の直前においてのみ「＜Ｎ＞」をつけて遅延時間を指定できる。

ｓｅｔ、ｒｅｓｅｔは、オペランドを次の括弧に囲むという形でのみ用いられ、それぞれ全ビットを１に、全ビットを０にするという意味で、単独に用いられる。これは、後述する、代入文と同時に動くことが許され、且つそれよりも優先される。ｄｃは、後述する、「並列動作の箇条書」において矢印「－＞」の直後に単独で用いられ、矢印の左の条件式を満たす入力変数の組は起こり得ず、禁止入力（ｄｏｎ’ｔ ｃａｒｅ）であることを表す。ｎｏｃは、後述する「同期信号に同期した動作」において、コロンイコール「：＝」の直後に単独で用いられ、左辺のレジスタ、メモリ、制御レジスタの内容は変化しないことを、陽に、表す。

4．ファシリティ

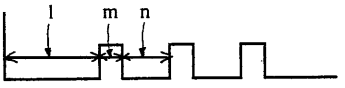
クロック、レジスタ、メモリ、組合せ論理回路、入出力双方向論理端子、バス、ユーザ定義のオペレータ、定数を「ファシリティ」と呼ぶ。

どのファシリティも、「：」と「；」の間に、「，」で区切って複数個列挙できる。

4－1．クロック

clock ： ＩＤ ＝ ｌ｜ｍ，ｎ；

の形式で、初期位相が「ｌ」、論理１レベルの期間が「ｍ」、論理０レベルの期間が「ｎ」単位時間であるようなクロック信号を表す。



「ｌ」は、A²DLにおいて唯一、負数とすることが許されており、これによって論理１レベルからはじまるクロックを宣言できる。

4－2．レジスタ

register：ＩＤ，
ＩＤ［ｍ］，
ＩＤ［ｍ：ｎ］，
ＩＤ［ｍ－ｎ］；

の形式によって、レジスタを宣言できる。添え字がない場合はＦ／Ｆ，添え字がひとつだけならこの添え字名の１ビットだけからなるＦ／Ｆ、［ｍ：ｎ］は第ｍビットからはじまり、ＬＳＢ方向へｎビット、［ｍ－ｎ］は第ｍビットからはじまり、第ｎビットでおわる｜ｍ－ｎ｜＋１ビットのレジスタである。

4－3．メモリ

memory：ＩＤ（ｉ：ｊ）＜ポート＞
ＩＤ（ｉ：ｊ）［ｎ］＜〃＞
ＩＤ（ｉ：ｊ）［ｎ：ｍ］＜〃＞
ＩＤ（ｉ：ｊ）［ｎ－ｍ］＜〃＞

の形式によってメモリを宣言できる。「ｉ」は開始番地、「ｊ」は語数、「ｎ」、「ｍ」の添え字のセマンティクスはレジスタの場合と同様である。シングルポートメモリであるなら、「ポート」には、番地を決定する、単なる「式」を書くだけでよい。マルチポートメモリの場合は、ポートのＩＤと、番地を定める式を「：」で結んだ組を、「，」で区切って列挙する。

4－4．組合せ論理回路

logic：ＩＤ ．．．＝ ．．．；

という形式によって組合せ論理回路を宣言できる。はじめの．．．の形式はレジスタの宣言と変わらない。あとの．．．には「式」を書く。ここでは、ｓｅｔ，ｒｅｓｅｔ，ｄｃ，ｎｏｃを除く、３５個のオペレータすべてを用いてよい。「＝」の左右でビット幅は、完全に、一致していなくてはならない。

4－5．入出力双方向論理端子

input ： ．．．；
output ： ．．．；
inout ： ．．．；

の形式によって入出力双方向論理端子を宣言できる。．．．の形式はレジスタの宣言と変わらない。

4－6．バス

net：ＩＤ ．．．＝｜ ＝＞ ．．．
｜＜＝ ．．．
｜＜＝＞ ．．．；

の形式によって、入出力双方向論理端子の接続関係を宣言できる。「＝」の左の「．．．」の

形式はレジスタの宣言と変わらない。２番目以降の「．．．」は、「ロケーション」と入出力双方向論理端子参照の組である。ここにロケーションとは「．．」と「ＩＤ」を「／」で任意個つなげたもので、「．．／」は「ひとつ上の階層の」、「ＩＤ／」は「ＩＤという名のモジュールの下の」という意味を表す。

４－７．ユーザ定義のオペレータ

operator：ＩＤ＜引数＞．．＜出力関数＞；

の形式によって、オペレータを宣言できる。「引数」の宣言は、レジスタの宣言と同じ形式をカンマで区切って行う。．．に何も書かなければ１ビット出力、他に「［ｍ］、［ｍ：ｎ］、［ｍ－ｎ］」を書くことができ、意味はレジスタの宣言と同じである。出力関数の部分には、組合せ論理回路の宣言と同様な記法を列挙する。全体として、引数から出力を定める組合せ論理回路になっていなくてはならない。オペレータは、「式」の中で、各引数に対し他の「式」を割り当てて用いる。

４－８．定数

constant：ＩＤ　＝　Ｎ；

はプリプロセッシング機能である。この宣言の行われたモジュール内の、「即値数」を書くことのできる任意の箇所で、「Ｎ」の代わりに、「ＩＤ」を用いることができる。

４－９．仮想ファシリティ

virtual：ＩＤ．．．＝　．．．；

の形式によって他のレジスタ、メモリの特定のポート、入出力双方向論理端子の一部あるいは全部のフィールドを別の名前で参照し、後述する代入文左辺や右辺で用いることができる。はじめの．．．の形式はレジスタの宣言の場合と変わらない。２番目の．．．では他のレジスタ、メモリの特定のポート、入出力双方向論理端子の「参照」を行う。「＝」の右に「式」を書くことは許されない。ビット幅は「＝」の左右で，完全に、一致していなくてはならない。

４－１０．宣言における添え字

以上述べた、ファシリティの「宣言」においては、添え字の流儀が、後述する、各モジュールに対するそれと異なることを禁止している。流儀の指定は、［ｍ：ｎ］のセマンティクスを定める手段だと解釈することもできる。

４－１１．参照における添え字

メモリのポート、組合せ論理回路、バス、オペレータの宣言におけるそれを含む、「式」一般において入出力のフィールドを指定する手段が「参照」である。「宣言」と「参照」では、添え字指定の意味が劇的に異なる。

「参照」において、添え字を指定しないことは、全フィールドの指定だと解釈される。モジュールに対する「ｌｓｂ＝０」という流儀指定のもとで、ｍ＜ｎな［ｍ－ｎ］の参照をしても、単にＬＳＢ側からＭＳＢ側に向かっての、「ｍ」という名のビット位置から「ｎ」という名のビット位置までの参照を意味するにすぎない。「ｍｓｂ＝０」の場合も同様である。

４－１２．モジュール間アクセス

記法「ｕｓｅ」によって、バス、オペレータ定数を除くファシリティや仮想ファシリティそして制御レジスタにアクセスすることを宣言できる。Ａ^{2}ＤＬには、ユーザ定義のオペレータを用いる場合以外に、引数の割当で結線を表現する手段は存在しない。アクセスしようとするファシリティの直前には、バスの表現におけるそれと同じ「ロケーション」を指定する。「ロケーション」の次にはもとの宣言をそのまま「引用」する。添え字［ｍ：ｎ］のセマンティクスは、直接アクセスの対象としているモジュールのそれに従う。「ｕｓｅ」での「引用」の内容はもとの宣言と完全に同一であることが要求される。ただし、「：」を用いているか、「－」を用いているかの相違は問題にされない。

制御レジスタに直接アクセスするには、

use state：「ﾛｹｰｼｮﾝ」「ﾓｼﾞｭｰﾙ名」；

の形式を用いる。

「＝」を用いて別名を用いることもできる。

５．代入文

代入文には、「：＝」を用いる「同期信号に同期した動作」と「＝」を用いる「端子との結線」の２種類がある。制御レジスタを対象とする「状態遷移」の表現には前者のみを許している。同一の制御レジスタあるいは同じファシリティの同じビット位置を対象とし、異なる代入文が同時に動作することは禁止している。

５－１．同期信号に同期した動作

「：＝」の左にレジスタ、メモリの特定のポートそしてこれらを対象とする仮想ファシリティの参照、右に式を書いて、このモジュールに

対して指定された同期信号の指定されたエッジ（前縁／後縁）に同期した代入動作を表す。また、「：」と「＝」の間に、モジュールに対する指定と同じ記法で、この代入文固有の同期信号とエッジを指定することもできる。この指定がある場合は、モジュールに対する指定よりも優先される。「：＝」の左右で、ビット幅は、完全に、一致していなくてはならない。右辺の式は指定された同期信号のエッジで評価され、その直後に、左辺で参照されたファシリティの出力側から、観測可能になる。

5－2．端子との結線

「＝」の左に、レジスタ、メモリの特定のポート、入出力双方向論理端子そしてこれらを対象とする仮想ファシリティの参照、右に式を書いて、左辺の参照内容と右辺で指定された式との直結を表す。「＝」の左右においても、ビット幅は、完全に、一致していなくてはならない。

「＝」は、この代入文に制御が渡されている期間全体にわたって、右辺の式の値が左辺のファシリティ、通常は論理端子、に供給されつづけ、出力側から観測され続けることを表す。特に、左辺がレジスタ、メモリいずれかの記憶素子だった場合は制御が他に移っても、その最後の値が保持されつづける。このことは「スルーラッチ」の表現に利用できる。左辺が入出力双方向論理端子の場合は、その新しい値を定める別の代入文がなければ、値は未定義になる。

5－3．状態遷移

「：＝」の左にモジュールを指定し、右にそのモジュールがとりうる状態のひとつを指定して、このモジュールに対して指定された同期信号の指定されたエッジ（前縁／後縁）に同期した状態遷移を表す。レジスタなどに対する場合と同様に、「：」と「＝」の間に、固有の同期信号とエッジを指定することもできる。左辺では、この文が書かれるモジュール自身であるなら、「ｓｔａｔｅ」という予約語のみを書いてそのことを示し、異なるモジュールであるなら、「ｓｔａｔｅ（ＩＤ）：＝」のようにモジュールの識別子（ＩＤ）を書いてそれを指定する。異なるモジュールの状態遷移を表現するには、記法ｕｓｅを用いて、予め、モジュール間アクセスを宣言していなくてはならない。指定されたモジュールが新しい状態になったことが観測可能になるのも、指定された同期信号の指定されたエッジの直後である。

右辺では、左辺で指定したモジュールがとりうる新しい状態を

ＩＤ

ＩＤ＾ＩＤ，ＩＤ

ＩＤ＾ＩＤ，ｃｌｅａｒ

のいずれかの記法で宣言する。状態遷移を指定しようとするモジュールの状態単位の動作が、後述する「ｓｅｃｔｏｒ」によって、セクタに分けられていないか、分けられていてもセクタ内の遷移ならば、右辺は「ＩＤ」のみによって状態名を指定するだけになる。セクタに分けられており、セクタ間の遷移を表現する場合は、「ＩＤ＾ＩＤ，ＩＤ」によって「セクタ名」、「そのセクタ内の状態名」、「戻り先の状態名」をこの順に指定する。「戻り先の状態名」は、必ず、この代入文が書かれたセクタ内の状態の名前でなくてはならない。それは状態遷移時にスタックにプッシュされる。スタックは深さ0からはじまる。「ＩＤ＾ＩＤ，ｃｌｅａｒ」は別のセクタに遷移する時、このスタックの内容をすべて初期化する記法である。この記法は、強制的なリセットや割り込みを表現する際に用いることができる。また「＝」のすぐあとに、「＜Ｎ＞」の記法で数字を書いて、スタックの任意の深さの内容を操作することもできる。このとき、モジュールの外見えの状態は変化しない。この「＜Ｎ＞」を用いた場合は、上記のうち「ＩＤ」と「ＩＤ＾ＩＤ，ｃｌｅａｒ」の2通りのみが指定できる。「ＩＤ」のみを用いると深さＮの状態のみが更新され、「ＩＤ＾ＩＤ，ｃｌｅａｒ」が用いられるとセクタと状態の両者が更新される。いずれの場合も「Ｎ」で指定した以外の深さの内容は変化しない。

「ｎｏｃ」（ｎｏ　ｃｈａｎｇｅ）を用いて、状態が変化しないことを陽に表現することもできる。他に、「ｒｅｔｕｒｎ」、「ｅｘｉｔ」が「：＝」なしの単独で用いられる。遷移したセクタの中で、ｒｅｔｕｒｎに出会うと上記のスタックの内容がポップされ、新しい状態となる。ｒｅｔｕｒｎに出会う前に別のセクタへの状態遷移文に出会うと、戻り先が続けてスタックにプッシュされ、遷移はネストすることになる。「ｅｘｉｔ」は未定義状態への遷移であり、

これ以降の動作は定義されないことを意味する。これらに対し、固有の同期信号を指定したい場合には、その直後でこれを行う。

6．並列動作（ＣＤＬ－ｌｉｋｅな記法）

「条件式」-)「代入文」,「代入文」・・；

の形式を「並列動作」とよび、条件が成立する場合、列挙された代入文のすべてが、一斉に、並列に動作することを表す。「条件式」は条件を表す任意の「式」である。条件を特に指定しない事も許され、この場合は、すべての代入文が常時動作することを表す。

「：＝」を用いる代入文は、左の条件が成立するなら、代入文左辺の記憶素子や制御レジスタが、同期信号の指定されたエッジの直後に、その直前に評価された右辺の内容に更新されることを表す。記憶素子では内容を交換できるが、「状態」を交換することは考えない。

「＝」を用いる代入文は、条件が成立している間、右辺で指定された式の内容が左辺のファシリティに供給され続け、このファシリティの出力側からも観測され続けることを意味する。

7．状態単位の動作（ＤＤＬ－ｌｉｋｅな記法）

```
「状態指定」：「並列動作」｜
              「並列動作」｜
                ．．．
              ；
```

の形式で、各状態毎に、たて棒「｜」で区切って並列動作を列挙することでコントロールフローが表現できる。「並列動作」は「条件式」を含んでもよいし含まなくてもよい。いずれの場合もそのモジュールがこの状態にあるとき、つまりこの状態に制御がわたされたときに活性化される。状態遷移を表す文が複数個活性化されることは禁止している。特に条件なしの状態遷移が書かれている場合は、これ以外に状態遷移を表す文を書くことは許されない。

7－1．状態のみを用いた記述

状態指定にＩＤのみを用いた場合は、各並列動作の条件に、さらに「その状態にあって」という条件が課され、「ｓｔａｔｅ：＝ＩＤ」のみで状態遷移が表現される。初めに書かれた状態が初期状態だと解釈される。

7－2．セクタを用いた記述

```
sector　ＩＤ：
      ＩＤ：「並列動作」｜．．．；
      ＩＤ：「並列動作」｜．．．；
                ・
sector　ＩＤ：
      ＩＤ：「並列動作」｜．．．；
      ＩＤ：「並列動作」｜．．．；
                ・
```

の形式で２組のＩＤを用いて、状態をさらに「セクタ」にまとめると、この各並列動作の条件には「そのセクタのその状態にあるか」という条件が課される。初めのセクタの初めの状態が初期状態だと解釈される。

7－3．待ち合わせ

```
「状態指定」：wait（「条件式」）
              「並列動作」｜
                ．．．；
```

の形式で条件式を指定し、この状態に遷移しても、条件式が満たされるまではどの並列動作も実行されないという「待ちあわせ」を表すことができる。セマンティクスはすべての並列動作に、共通に、この条件をＡＮＤで含めた場合とまったく変わらない。

7－4．状態の参照

オートマトンの状態は、

ＩＤ	どの状態にあるか
ＩＤ＾ＩＤ	どのセクタのどの状態にあるか
ＩＤ＾	どのセクタにあるか

の３通り、そして、スタックの深さＮの内容は

ＩＤ＾ＩＤ	どのセクタのどの状態がしまわれているか
ＩＤ＾	どのセクタがしまわれているか

の２通りで参照できる。自分自身の状態は

```
state　eq　ＩＤ
state　ne　ＩＤ＾ＩＤ
```

のように「ｓｔａｔｅ」のみで参照され、他のモジュールのそれは

```
state（ＩＤ）　eq　ＩＤ
state（ＩＤ）　ne　ＩＤ＾ＩＤ
```

のように参照される。深さは「ｅｑ」、「ｎｅ」の直前に「＜Ｎ＞」で指定する。他のモジュールの状態を参照するには、状態を外部から強制的に遷移させる代入文の場合と同様に、モジュール間アクセスを宣言しておく必要がある。

7－5．衝突の回避

```
escape　「条件式」　；
```

の形式を、１モジュールに１度だけ用いて、

「以下の条件文にはすべて『この条件式が成立せず且つ』が略されている」ことを指定できる。外部からは、この条件が成立しているときにのみ、このモジュールに強制的に何かをさせることにすれば衝突は必ず避けられる。

8．階層構造の表現

A²DLにおいて階層性は以下で述べるモジュールを入れ子の形で宣言することで表現される。

8－1．モジュールの宣言

module ID:「流儀指定」:「同期信号指定」:
「ファシリティの宣言」
「（下位の）モジュールの宣言」
「並列動作の箇条書」
「状態単位の動作の箇条書」 ；

の形式でモジュールを宣言できる。

はじめの「ＩＤ」がモジュール名を表す。「流儀指定」では添え字の流儀が「ｌｓｂ＝０」（降順）、「ｍｓｂ＝０」（昇順）のいずれであるかを指定する。「同期信号指定」では

「ﾛｹｰｼｮﾝ」「同期信号」, positive

のように、このモジュールの「：＝」が同期する１ビット出力の式を指定する。「ロケーション」はこのモジュール内の宣言がベースで、このモジュール内で宣言されたクロック等を指定する場合は不要である。「ｐｏｓｉｔｉｖｅ」は前縁を表し「ｎｅｇａｔｉｖｅ」は後縁を表す。それぞれ「ｐ」、「ｎ」と略せる。

「ファシリティの宣言」部分ではクロック、レジスタ、メモリ等のファシリティを所定の形式で列挙する。「（下位の）モジュールの宣言」では、下の階層にモジュールがあるならば、その全体を入れ込んだ形で表現する。下位にモジュールがなければ空になる。「並列動作の箇条書」の部分には、常時動き続ける代入文を、必要なら条件を添えて、箇条書する。「状態単位の動作の箇条書」のところには、状態単位の動作を表現する。「並列動作の箇条書」と「状態単位の動作の箇条書」はいずれも空でありうる。

特に最上位のモジュールには「ｍｏｄｕｌｅ」の代わりに「ｍａｉｎ」という予約語を用いる。

記述は、「ロケーション」をはじめに添えて、モジュール単位に別ファイルに格納できる。

8－2．モジュールの複製

duplicate「ﾛｹｰｼｮﾝ」ＩＤ１－＞ＩＤ２ ；

という記法で「ＩＤ１」とまったく同一のモジュールを「ＩＤ２」という名前で、この記述をしたロケーションに複製できる。同一のロケーションのモジュールを複製の対象としているなら「ロケーション」はなくてよい。

9．あとがき

本稿では、マイクロアーキテクチャ評価システムに関する研究成果[8]を基に、ＡＳＩＣを多用する論理装置の、デバイステクノロジーに依存しない範囲の設計上流工程完全自動化を指向して開発を進めた自動設計システムの核言語：A²DLの全容を紹介した。筆者らが想定している設計に関するかぎり記述能力にはなんの問題もなく、きわめて高い完成度を有すると自負している。すでにA²DLを入力してシミュレーションと、コントロールフローを破壊しない範囲の、シンセシス行う高レベル自動設計システムA²DL－DAの初版を実現し[4][7]、５品種のＬＳＩ設計に適用して、言語仕様と処理系の基本機能／性能を実証済みである。現在は、実運用にむけて、合成能力の強化を行っている。

文献

（１）S.G.Shiva:"Computer Hardware Description Language-A Tutorial",Proc. IEEE, vol.67,No.12 (1979)

（２）J.R.Dulay,D.L.Dietmeyer:"A Digital System Design Language(DDL)",IEEE Trans. Comp. Vol.C-17,No.9 (1968)

（３）高橋、村岡、林：「自動方式・機能設計システムA²DL－DA」、第３８回情処全大４Ｓ－２（１９８９年）

（４）高橋：「A²DL－DAの構成と実現」第４０回情処全大３Ｍ－５（１９９０年）

（５）村岡、赤川、江尻、林：「A²DL－DAを用いた論理装置用ＬＳＩの上流設計」第４０回情処全大３Ｍ－６（１９９０年）

（６）高橋：「ブール式を要求駆動で評価するＲＴレベルシミュレーション」情処研報Ｖｏｌ．９０，Ｎｏ．１４（１９９０年）

（７）高橋：「大規模化、高度化への対応に不可欠な高レベル設計自動化」、ＣＡＤ＆ＣＩＭ、第４巻第６号、工業調査会、（１９９０年）

（８）R.Takahashi,T.Yoshimura,S.Goto:"A VLSI Architecture Evaluation Sysytem" Proc.ICCD'86 (1986)