設 計 自 動 化 22－1
（１９８４．７．１７）

階層的ハードウェア設計言語Ｈ２ＤＬの思想

宮田 操　　西尾 誠一　　山崎 勇

東芝 総合研究所

§１ はじめに

半導体技術の目覚ましい進歩により、１００Ｋゲートにも及ぶ論理ＶＬＳＩの製造が可能となり、まさに“system on a chip”の時代に突入したと言える。このように益々複雑・大規模化する論理回路の設計に対処するために、各種の論理シミュレータや自動レイアウト・プログラムの開発が精力的に進められて来ており、これらのＣＡＤプログラムを用いることによって、ゲート・レベルからマスク・パターン・レベルまでの言わば物理的な設計過程に必要な時間は大幅に短縮されつつある。しかしながらその結果として、仕様からゲート・レベルまでの機能・論理設計の時間が、トータルの設計時間のうちで大きなウェートを占めるようになって来ている。それ故、ＶＬＳＩのような大規模論理回路の設計効率を一段と向上させるためには、設計の上流過程である機能・論理設計の時間を短縮することが必要である。

機能・論理設計の効率向上には、階層的設計手法の導入が唯一の解と思われる。この階層的設計手法自体の考え方は目新しいものではなく、この手法に基づくハードウェア設計言語［１］も従来からいくつか提案されている。

これら言語を用いて論理回路を設計する場合は、まず回路（モジュール）の構成要素（サブモジュール）の種類とその仕様、すなわち各サブモジュールの機能と入出力信号の数・意味等を定めたうえで、それらの間の接続関係を記述する。これで設計が一段階進んだことになり、次いで個々のサブモジュールを同様に順次設計して行くことにより、階層的に設計が行われることになっている。

しかしながら、全体を一度に見通し難い大規模論理回路の設計では、サブモジュールの詳細な仕様を設計が完了する前に決定することは不可能に近く、上述のように、“接続記述を中心にして階層的に設計を進める”ことを基本としているハードウェア設計言語を大規模論理回路の設計に直接適用することはできない。

筆者らはこのような考え方から、機能・論理設計レベルで階層的設計手法を効率よくサポートできることを目指した階層的ハードウェア設計言語Ｈ２ＤＬ（Hierarchical Hardware Design Language）の開発を進めている［２］。

以下本文ではＨ２ＤＬの基本的な考え方と主な特徴点、及びＨ２ＤＬを用いた階層設計のプロセスについて述べる。なおＨ２ＤＬの言語仕様面については触れないので、これについては本研究会資料２２－２［３］を参照して欲しい。

§２ Ｈ２ＤＬの基本的な考え方

従来提案されているハードウェア設計言語を用いて機能レベルから論理回路を階層的に設計する場合、設計者は次のような矛盾に直面する：

（ａ） サブモジュールの機能はもちろんのこと、入出力信号の種類・意味までを全て明確に決めた後、設計対象であるモジュールをサブモジュールの接続関係で表わす。

（ｂ） サブモジュールの入出力信号の数等は、モジュールの設計が全て完了しなければ明確には決まらない。

この矛盾は通常の設計過程では、設計者が頭の中でモジュールの選定を試行錯誤で繰返すことにより、解決されていると考えられる。しかしながら、ＶＬＳＩのような大規模論理回路の設計をこのようなスタイルで行っては、時間がかかって効率が悪いだけでなく、収拾がつかない場合もありうる。

この問題に対する解決案として、筆者らは次のような基本的アイディアに基づいて、Ｈ２ＤＬの開発を始めた。

（ａ´） サブモジュールの機能だけを前もって決めるのは、入出力信号の数・意味までを決めるのに比べて、さほど困難ではない。

（ｂ´） サブモジュールの入出力信号等を決めるのは、そのサブモジュールの使われ方をよく調べた上でのほうが楽であり、またモジュール全体としても効率のよいものが決められる。

その結果Ｈ２ＤＬではサブモジュールとして機能だけは定まっているが、制御信号等入出力信号の種類・意味は未定のままにしている“ハードウェア・プロセス“と呼ぶ概念を導入し、これを用いてまず最初に設計対象ハードウェアの機能記述を行う方法を採用した。ハードウェア・プロセスとして取扱われるサブモジュールの制御信号は、この機能記述を整理して得られる情報［５］を参照して適当に定められ、その後であらためて全体の接続関係が記述される。すなわちＨ２ＤＬを用いた設計手

順によると、各サブモジュールは設計の最初の段階では機能だけに着目したハードウェア・プロセスとして取扱われ、その後全体の設計が進みモジュールの機能記述が完了した時点で、具体的な仕様が決定されて接続関係が記述される。次いで同様な手法で個々のサブモジュールの設計に進むことにより、階層的に設計を行うことが可能になっている。このためH2DLはSDL等に対比して、“機能記述を中心に据えた階層的ハードウェア設計言語”と言うことができる。

§3 H2DLの構成

前節で述べた考え方に基づいたH2DLは、内部仕様記述、接続記述、ハードウェア・プロセス記述及びインタフェース記述の4種類の記述形式から成る。

(1) 内部仕様記述

内部仕様記述では、モジュールの動作内容がそれを構成するサブモジュールを用いて記述される。基本的な記述の形式は

if 条件 then 回路動作

であり、レジスタ・トランスファ・レベルの言語の範疇に入る。しかしながらDDL[6]等従来の同レベルの言語と大きく異なり、サブモジュールとして使えるものがレジスタ、ラッチ、メモリ等あらかじめシステムに登録されているものに限られていない。

H2DLでは、設計者は任意のサブモジュールをハードウェア・プロセスとして定義し、これらを使ってモジュールの動作を記述することができるようになっている。

H2DL内部仕様記述では、ハードウェア・プロセスをモジュールの内部に組込んだ形で全体の動作記述がなされるため、サブモジュールの入出力信号をこの段階で決めておく必要が無いが、DDLで用意されている“エレメント”のようなblack box を使う場合は、そのblack box を外部に出しモジュールの残りの部分の動作を記述することになるため、必然的にblack box として取扱うサブモジュールと残りの部分との入出力信号を決めねばならない。

またこのハードウェア・プロセスはサブモジュールの動作を定義しているため、black box とは異なり内部仕様記述されたモジュールの動作を正確にシミュレーションすることも可能である。

なおH2DLではレジスタ、メモリ等も含め、内部仕様記述に現われるサブモジュールは、全てハードウェア・プロセスとして統一的に扱われる。

(2) ハードウェア・プロセス記述

ハードウェア・プロセスの概念は、サブモジュールの入出力信号等の決定はできるだけ遅らし、当初は機能面だけを考えて階層設計を進められるようにするために導入されたものである[4]。

ハードウェア・プロセスはサブモジュールの機能だけに着目し、入出力信号や制御信号等は一切考慮しない論理回路の一抽象化表現と言える。このハードウェア・プロセスは複数の機能を持つことが可能であり、各々の機能はプロセスとして定義される。

ここでプロセスとは、例えばALUをハードウェア・プロセスとして考えたときのADD機能、SUB機能等が相当している。

論理回路の抽象化には、機能自体の抽象化のほかに、機能を実行させるための制御方法、入力データの与え方及び出力データの取出し方についての抽象化が考えられる。このうち後者の抽象化はH2DLでは次のようになされている。

内部仕様記述での各機能の実行は、対応するプロセスを次の形で呼出す(プロセス・コール)ことで指定される[制御方法の抽象化]。

HP. P(AG1, AG2, ・・) AT T (*)

ここでHPはハードウェア・プロセスの実体名、Pは実行したい機能に付けられたプロセス名を表す。またプロセスの実行に必要な入力データは、プロセス・コールの引き数AG1, AG2・・として与えられる[入力データの抽象化]。

このようにH2DLでは、単にハードウェア・プロセス実体名とプロセス名を指定するだけで任意の機能を使うことができるため、細かな制御方法や入力データの渡し方などは、内部仕様記述の段階では決めておく必要がない。

またプロセスの実行結果も、ハードウェア・プロセス実体名を指定するだけで参照することができる[出力データの抽象化]。従って入力データだけでなく、出力データについてもそのデータ・ポートの構成を気にすることはない。

プロセスの構造－機能の抽象化－

H2DLでは各プロセスは、Cパート(Combinational Logic Part)とSパート(Sequential Logic Part)の二種類より成るとし、その構造として第1図に示すようなハードウェア・モデルを仮定している。ここでCパートは組合せ回路であり、Sパートは組合せ回路と順序回路から構成されている。

このモデルは原理的には任意の論理回路の動作を表わすことができるものであるが、H2DLではモデルの実行タイミングに次のような制限を付加している。

■ C、Sパートの実行タイミングは(*)の信号Tによって指定される。すなわち、図2に示すごとく、Cパートは信号Tが0の間中常に実行・評価され、Sパート

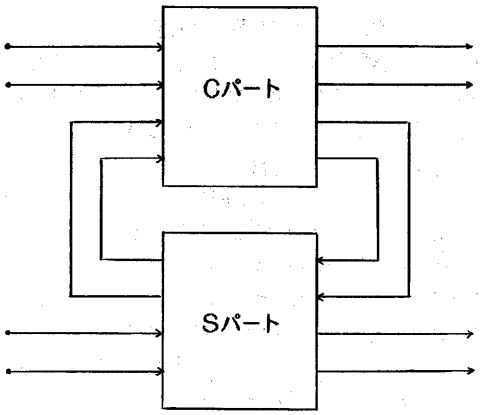

図1　プロセスのモデル

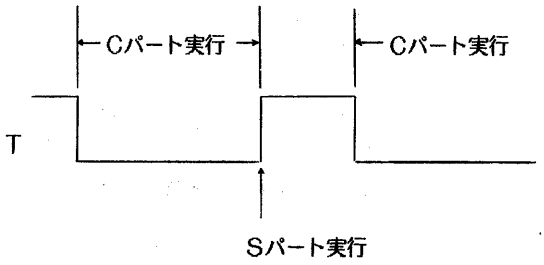

図2　プロセス実行のタイミング

は T の立上り、つまり 0 から 1 に変化した瞬間に実行されるとする。従って T が 0 の間にもしプロセス・コールの引き数が変化すれば、プロセスの実行結果もそれに伴って変化することになる。またプロセスの状態変化は、S パートの実行によってのみ引き起される。

このように C、S パートの実行タイミングを定めたことは、モジュールを次のようにモデル化したことに相当する。

■　モジュールの各機能を 2 つのサブ機能に分ける。一方（C パート）は組合せ回路的で、ある期間中実行され、他方のサブ機能（S パート）は状態変化を伴うもので、本来の実行時間を言わば一点に凝縮した形で瞬間的に実行されるとする。

このようなモデル化では、もちろん全ての論理回路の動作を記述することはできないが、

（a）　モデルが簡単で、直観的に理解し易く、また記述も容易である、

（b）　ハードウェア・プロセスはクロック単位での動作記述を基本としている内部仕様記述で用いられるため、通常は変化を起すタイミングを一点だけ指定できればよい、

等の理由から、H2DL で必要な論理回路の機能抽象化としては最低限満足できると考えている。

ハードウェア・プロセスの例

論理回路の機能をハードウェア・プロセスとして抽象化する具体例として、レジスタをとりあげてみる。機能抽象化は次のように考えることができる。

（1）まずレジスタの基本的性質として記憶要素 STORAGE と、それに値をセットするプロセス IN を定義する。（2）もしこのレジスタが非同期クリア機能を有するならば、これに対応して STORAGE の値を 0 にするようなプロセス CLEAR を考え、その動作を記述する。（3）また STORAGE の値をカウント・アップできるのならば、その機能もプロセス CUP として定義する。（4）同様にして、必要な他の機能もそれぞれ適当なプロセスとして定義・記述することになる。

この様子を図3に示す。ハードウェア・プロセスとしてのレジスタは記憶要素 STORAGE と、機能が明確に定義されている複数のプロセス IN、CLEAR、CUP 等から成っているが、その制御信号とかデータ入出力端子、クロック端子、更にはセット・アップ・タイム、ホールド・タイム等はまだ未定のままである。

なおハードウェア・プロセスの記述上は、この記憶要素 STORAGE は各プロセスで自由にアクセスできる変数として定義される。

H2DL ではこのように未定部分があっても、設計者は内部仕様記述でこのレジスタの全ての機能を、単に対応するプロセスを（＊）で示す形式で呼出すことにより、自由に使うことができる。

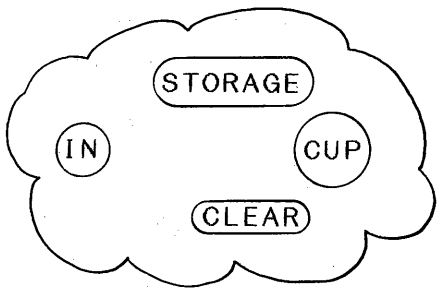

図3　ハードウェア・プロセスとしてのレジスタ

従ってプロセス・コールとは論理回路に所望の機能を実行させるための操作、と位置付けることができ、このプロセス・コールの手順が設計が進むにつれて次第に詳細化されていく。

例えば先ほどのレジスタで考えると、設計の初期では次のプロセス・コール

REG. CLEAR ()

でレジスタREGを非同期クリアするわけであるが、設計が進んでREGの制御信号が定まった時点では、同じ非同期クリアを記述するにも

CLEAR−REG=1

と言うように、クリア用制御信号、CLEAR−REGにexplicitに1をセットすることになる。

ハードウェア・プロセスはまた次のように考えることができる。DDL等の従来の機能記述言語でもサブモジュールとしてレジスタ、メモリ等が用意されており、それらを記述で使用する場合は、例えばレジスタを考えると、入力データとその入力条件、入力のタイミングを指定するだけで格別そのために必要なエネーブル信号の設定等は意識しないで済むようになっている。これはレジスタを抽象的にとらえ、データを入出力できると言う性質だけに着目して、そのために実際は必要な制御信号等は切捨てたものと考えることができる。DDL等はレジスタ・トランスファ・レベルの言語であるため、レジスタ、メモリ等だけにこのような配慮をしておけば充分であったが、H2DLは階層設計を有効にサポートすることを目指しているため、サブモジュールとしてレジスタ、メモリだけでは不充分であり、任意のものに対する対応策が必要である。このために導入された概念がハードウェア・プロセスであり、これはDDL等におけるレジスタやラッチの抽象化概念を一般の論理回路にまで拡張したものと言える。

ハードウェア・プロセスはこのように論理回路の機能だけに着目したものであり、また瞬間的に実行されるとモデル化しているため、その記述用言語としては通常のプログラミング言語と同じく手続き型のものを採用している。

なお、レジスタ，ラッチ，メモリ等もハードウェア・プロセスとして取扱うわけであるが、内部仕様記述でこれらにデータを転送する場合は特例として、プロセス・コールの形式ではなく通常の記述形式が用いられる。例えばレジスタREGへサブモジュールXの値を転送する場合、プロセス・コールの形式で書けば、

REG. IN (X)

となるが、これは従来からの形式で

REG=X

と書くことができる。

またレジスタ，ラッチに付属する機能として、クリア，プリセット，非同期ロード，カウント・アップ／ダウン等のプロセスがあらかじめ用意されており、設計者は自由に使用することができるようになっている。

（3） インタフェース記述

ハードウェア・プロセスは論理回路のうち、純粋に論理的な動作に関するものであり、実際にどのようにしてその機能を使うのかとか、データの入出力方法等については触れないものである。それに対して、制御信号の意味とかデータ・ポートの構成、更にはタイミング等を記述するものがインタフェース記述である。この記述では、制御信号がどのような値の時にどのプロセスが起動されるのか、またそのプロセスに必要なデータはどのような経路から入力されるのか、プロセスの実行時間はどの位か、といった事項が規定される。

インタフェース記述は基本的に1つのプロセスに対し、次の4項目が指定される。

プロセス名
プロセス実行条件
入力データ
タイミング

ここでプロセス実行条件は、制御信号やクロック信号間の特定な波形関係を記述するもので、この関係が満された時に対応するプロセスの実行が開始されることを示している。

ところでハードウェア・プロセスは各プロセスの論理的な動作内容を記述しているだけで、実際のインプリメンテーションには触れないものである。またインタフェース記述もプロセスと制御信号のパターンとの対応関係を示すだけであり、同様にインプリメンテーションについては関与していない。それ故、H2DLでは両者の記述を合せたものが、インプリメンテーションを考慮しない論理回路の“外部仕様”に相当することになる。

また同一のハードウェア・プロセスに対して、インタフェース記述を変えることにより、機能は同じであるが制御信号やデータの入出力方法が異なる論理回路を表わすことが可能である。

（4） 接続記述

接続記述では、設計対象のモジュールの構成をサブモジュール間の接続関係として表わす。

一般に接続記述においては、素子形式とネット形式とがあるが、H2DLでは素子形式を採用しており、以下のような形式で記述する。

サブモジュール識別名（出力並び）
＝タイプ名（入力並び）

ここでタイプ名はサブモジュールの定義自体につけられた名前であり、サブモジュール識別名は各サブモジュ

ールの実体名である。また出力並びではサブモジュールの出力端子に接続される信号の並びを、入力並びではサブモジュールの入力端子に接続される信号の並びを記述する。なお信号の記述ではビット幅を持つものも扱えるようにしてある。

（５）　４つの記述形式の相互関係

Ｈ２ＤＬの４つの記述形式を簡単にまとめると次のようになる。

内部仕様記述はハードウェア・プロセス記述されたサブモジュールを使って、設計対象モジュールの動作を“レジスタ・トランスファ・レベル”で記述するものであり、ハードウェアとしての具体的実現方法を意識している記述である。

ハードウェア・プロセス記述はモジュールが果す動作の実現方法には触れずに、純粋に機能だけをプロセスとして定義したものであり、モジュールの入出力信号とこれらプロセスとの関係、すなわちプロセスの実行条件や入力データのプロセスへの渡し方等を規定するものがインタフェース記述である。なお、ハードウェア・プロセス記述とインタェース記述を組合わせたものが、モジュールの外部仕様に相当する。

接続記述は具体的な論理回路に１対１に対応するもので、モジュールを構成するサブモジュール間の接続関係を示している。サブモジュールとしては、内部仕様記述や接続記述もしくはインタフェース記述＋ハードウェア・プロセス記述されているものが使用できる。

以上の４記述のうち、内部仕様、インタフェース及び接続の３記述では、記述対象であるモジュールの入出力信号が全て決まっている必要があるが、ハードウェア・プロセス記述ではこれらの信号について何ら考慮しない。

４つの記述形式の関係を図４に示す。図の内部仕様記述、接続記述及びインタフェース記述＋ハードウェア・プロセス記述は、同一の論理回路をそれぞれ別の観点から見た記述になっている。

なお外部仕様に基づき内部仕様記述を作成するのは当面、設計者に委ねるが、内部仕様記述から接続記述を生成するのは、自動論理合成プログラムによる予定である。

§４　Ｈ２ＤＬを用いた階層設計

論理回路を階層的に設計するスタイルには、トップ・ダウン方式とボトム・アップ方式の２種類がある。Ｈ２ＤＬは両者ともサポートできるものであるが、ＶＬＳＩのごとく大規模回路の設計に適していると思われるトップ・ダウン方式の側により重点をおいている。

トップ・ダウン設計

Ｈ２ＤＬを使うことにより、トップ・ダウン設計がど

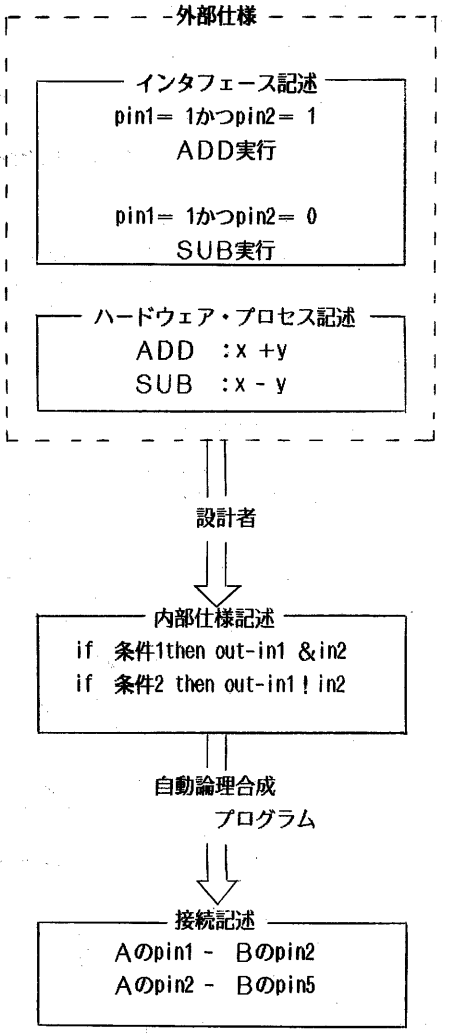

図４　記述の相互関係

のように進んで行くか、また各フェーズでＨ２ＤＬのどの記述が用いられるか、図５を参照しながら説明する。

（１）　まず設計対象である論理回路Ｘが果すべき動作や入出力信号の数・意味等が明確に規定された仕様書が与えられる必要がある。この仕様書は日本語の場合もあるが、Ｈ２ＤＬではインタフェース記述＋ハードウェア・プロセス記述として書かれる。

（２）　次いで設計者は（１）の仕様書に基づき、必要なサブモジュールを考えるとともに、データの経路や制御方法を検討して、回路Ｘの構成をＨ２ＤＬ内部仕様記述する。ここではサブモジュールＡ、Ｂ、Ｃを使うとする。内部仕様記述では、全てのサブモジュールはハードウェア・プロセスとして取扱われるため、設計者はこれ

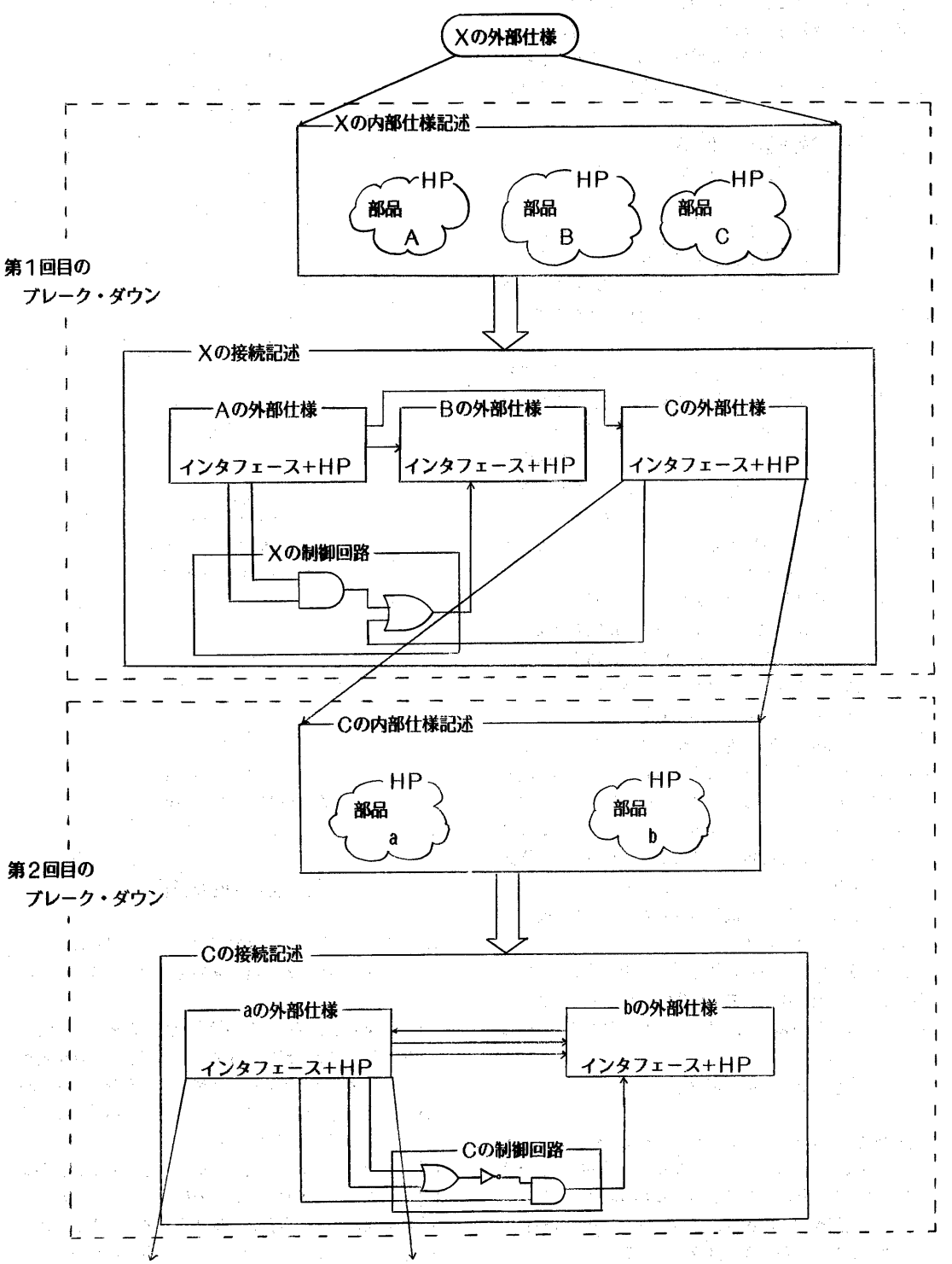

図5　H2DLを用いた階層設計

らA、B、CをモジュールXの仕様を満す上で必要な機能面だけに着目して選択することが可能である。

また内部仕様記述は基本的に

if 条件 then 回路動作

の形式であり、制御部の構成がexplicitにはあらわれない。

なお次のステップに進む前に、内部仕様記述と（1）の外部仕様との等価性をチェックしておく必要がある。

（2） 内部仕様記述が完成すると、各サブモジュール単位にその動作条件やタイミング等を調べる作業を行う［5］。これにより、ハードウェア・プロセスとして取扱ってきたサブモジュールの入出力信号に対する要件が明らかになるので、それに基づいて各サブモジュールのインタフェース記述を作成することができる。すなわちこの時点でサブモジュールのハードウェア・プロセス記述とインタフェース記述が揃い、その外部仕様が決まることになる。

更に各サブモジュールの入出力信号が決まると、モジュールの制御回路の仕様、すなわち何時どのような信号をどのサブモジュールに送ればよいかと行った事項も決まってくるので、これを実現する具体的回路をゲートで構成することになる。

（4） 以上が終了すると、サブモジュールA、B、Cと制御回路の間を、（1）で指定された仕様を満すように結線する。この結線関係はH2DL接続記述として書かれる。

なお接続記述と（1）の仕様、もしくは（2）の内部仕様記述とがもし等価でない場合は、接続記述自体に誤りがあるか、（3）で決めたサブモジュールのインタフェース記述に誤りがあったかのいずれかである。なぜならば、サブモジュールの機能はハードウェア・プロセスで代表され、それは既にステップ（2）で検証済みであるからである。

（5） 以上で最初与えられた仕様の論理回路Xが、サブモジュールA、B、Cと、Xの制御回路とに1段階、ブレーク・ダウンされたことになる。ここでサブモジュールはそれぞれインタフェース記述とハードウェア・プロセス記述を有している、すなわち外部仕様が明確に定義されているので、その仕様に基づき次の段階では、これらのサブモジュールの設計を行なうことになる。なお制御回路は既にゲート・レベルになっているため、これ以上のブレーク・ダウンは不要である。

サブモジュールA、B、Cの設計はステップ（1）から（5）を同様に繰返して進められる。このシーケンスは全てのサブモジュールが既設計のものか、あるいはゲート・レベルで直接構成できるようになるまで続けられる。

このようにH2DLを使うと、ハードウェア・プロセス記述、インタフェース記述、接続記述、内部仕様記述の繰返しにより、トップ・ダウン設計が進められて行くことになる。

トップ・ダウン設計の変形

完全なトップ・ダウン設計ではなく、一部のサブモジュールとして既設計のものを使うケースも多々ありうる。このような場合は既設計サブモジュールの入出力信号は明らかであるため、このサブモジュールと設計対象モジュールの残りの部分（モジュール´）との間の結線をまずH2DLで記述し、その後モジュール´を前述した手順でトップ・ダウンに設計して行けばよい。

§5 おわりに

階層的ハードウェア設計言語H2DLの基本的な考え方とその構成について述べた。H2DLは4つの記述形式により、“機能記述を中心に据えて”、階層的に大規模論理回路の設計を行なえることを目指している。特に論理回路の機能抽象化表現であるハードウェア・プロセスの概念を導入したことにより、サブモジュールの機能だけに着目して階層的記述を機能レベルからおこなえるため、今後益々高集積化して行くVLSIの設計用言語として適当であると考えられる。しかしながらハードウェア・プロセスの記述能力は必ずしも充分ではなく、今後はその抽象化能力の拡張も検討して行きたい。

なお本文ではH2DLの言語面の特徴についてだけ述べたが、現在この言語に基づくCADシステムも開発中である。主なプログラムには次のようなものがある。

（1） 機能シミュレータ

内部仕様記述で書かれた論理回路をシミュレーションし、選択したサブモジュールの機能やそれらの制御方法が妥当か否かチェックする。

（2） ミックスト・レベル・シミュレータ

H2DL接続記述された論理回路をシミュレーションするもので、各サブモジュール及び制御回路間の結線やサブモジュールのインタフェース記述の正しさをチェックする。サブモジュールとしては、内部仕様記述、接続記述もしくはインタフェース記述＋ハードウェア・プロセス記述されたものが使用できる。

（3） 自動論理合成プログラム

内部仕様記述を入力し、ゲート・レベルの制御回路の生成とサブモジュール及び制御回路間の結線関係を自動的に作成する。ただしサブモジュールのインタェース記述は設計者が指定することを

想定している。

このH²DLに基づくCADシステムについては、別の機会に発表したい。

参考文献

[1] van Cleemput, W. M., 1977, A Hierarchical Language for the Structured Description of Digital Systems, Proc. 14th DA Conf., 377-385.

[2] Miyata, M. et al, 1984, A Hierarchical Hardware Description Language, IEC EDA84, 189-193.

[3] 西尾他, 1984, 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－, 設計自動化研究会資料22-2.

[4] 宮田他, 1984, 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.

[5] 増渕他, 1984, H²DLトランスレータの概要, 同上, 2P-6.

[6] Duiey, J. R, and Dietmeyer, D. L., 1984, A Digital System Design Language (DDL), IEEE Trans. on Computers, C-17, no. 9, 850-861.