ソフトウェア基礎論　２５－７
（1988. 5. 26）

代数的手法を用いた順序回路の記述とその詳細化について

On Specification and Refinement of Sequential Logic Circuits by Algebraic Method

杉山　裕二　Y. Sugiyama
北道　淳司　J. Kitamichi
谷口　健一　K. Taniguchi

大阪大学基礎工学部

Faculty of Engineering Science, Osaka University

あらまし　代数的仕様記述法に基づいた同期式順序回路の仕様記述の一スタイル，実現の定義ならびに段階的詳細化手法を提案する．この方法では，論理回路の機能を状態遷移関数と，状態の持つ情報を取り出す出力関数とで定義し，一般に複数個ある状態遷移関数の実行順を bool関数 VALID を用いて記述する．状態 s で遷移 t を実行すべきとき VALID(t(s)) が真，そうでないとき偽となるように書く．回路の機能定義に相当する抽象的なレベルから，IC などの入力線の論理までこのスタイルで記述することができ，一方が他方の実現になっていることなどの証明も，検証支援系を利用して厳密に行えるなどの特徴がある．CPU の例題を用いて，具体的な記述法や詳細化法について述べている．

Abstract　Based on algebraic specification method, a style of writing specifications of syncronous sequential logic circuits, a definition of implementation, and a top down refinement method for the specification are presented. A logic circuit is described by defining state-transition functions and output functions. A valid next state-transition is specified by defining a function "VALID". We can write specifications of a logic circuit on various levels of abstraction from a level of function definition to that of logic design in the same style. Taking a micro-computer as an exsample , we illustrate how to write a specification in a abstract level and how to refine the specification to a realization level in which input logic of modules (e.g. register, counter, ALU) is described.

1．まえがき

ソフトウェアやハードウェアの仕様記述において重要なことは，（イ）抽象度の高いレベルから具体的なレベル（ソフトウェアの場合はコンパイラがあるプログラムのレベル，ハードウェアの場合は標準的なICを用いた論理設計（入力論理式の記述）のレベル）までを含め，任意の抽象レベルで書きたいことだけを自由に記述できること，（ロ）言語の意味定義や実現（詳細化）の正しさの定義などが厳密・形式的かつ明快であること，等であろう．実現の正しさの検証（ここでは厳密な意味での証明）を行うためには，さらに（ハ）全てのレベルの記述が同一言語で行われること，も重要であろう．代数的言語はそのような特徴を持ち合わせていると考えられる．プログラムの仕様記述や作成に関する代数的手法の研究は比較的行われているが，ハードウェアの設計に関しては数が少ない[1,2,3]．筆者らは，ハードウェア特に同期式順序回路を対象にして，筆者らのグループが設計した代数的言語ＡＳＬを用いて，順序回路の仕様をどう書くか，実現（詳細化）の定義をどうするか，抽象レベルから具体レベルまで記述できるか，書きたいこと（例えばこういうレジスタ・バス構成にしてこれとこれのデータ転送は同時に行いたいなど）が自然に記述できるか，正しさの検証はどのように行うか，設計で自動化できる部分はあるか，それをどのように行うか，等について検討してきたので，その一部を報告する．

この報告ではまず，代数的手法を同期式順序回路を中心としたハードウェアの仕様記述に適用した場合の記述のスタイルや実現について論じる．同期式順序回

路は，クロックなどの入力により，状態遷移が次々に起こり，それに伴い内部のレジスタなどの値が変化していくものとみなすことができる．そこで，回路記述の状態遷移の意味定義に関する部分を抽象的順序機械[4]のスタイルで行う．まず，回路の抽象的な全状態を表すデータタイプを導入し，それを引数とする状態遷移関数及び出力関数を設定する．そして，各状態遷移後の出力関数の値を，状態遷移前の出力関数の値の複合関数として定義することにより仕様記述を行う．この抽象状態は，回路に含ませるメモリや，レジスタ，カウンタ，フリップフロップ等の全内容を抽象的に表すものである．メモリ内の一命令実行とか，レジスタ間のデータ転送などが１つの状態遷移と見なせる．任意の抽象レベルにおいても，そのレベルでの出力関数と状態遷移の意味がこのスタイルで記述できる．

ソフトウェアあるいはプログラムの仕様記述においては，“実行系”があって，項の値（例えば，初期状態にこれこれの状態遷移関数をある条件に従って演算して得られる状態におけるある出力関数の値）を計算するとき，どの順に計算するかとか，今までの結果を引数にして次にどの関数を計算するかなどは実行系が決めてくれる．しかし，ハードウェアの同期式順序回路ではクロックが供給されるごとに，次にどの状態遷移を実行すべきかは回路自身が決めなくてはいけない．本報告では，状態遷移の実行順を記述するために，述語記号VALID（VALID関数と呼ぶ）を導入し，状態 s で遷移 T を実行すべきとき，VALID(T(s,…))が真，そうでないとき偽となるように書くことにする．従って，この抽象状態sの中には，通常の順序回路でいういわゆる「状態」（有限制御部の「状態」）がどの「状態」にあるかという情報ももっていなければならない．一連の指定した状態遷移のみを行うように記述するには，一般に，有限制御部の状態が今何かを表すような新たな出力関数（例えばcontrol_state(•)）を導入する必要がある．各状態遷移Tについて，今Tを実行すべきかどうかを，control_state値も含め全ての出力関数値を用いた論理式で書き表す．しかし，筆者らは，有限制御部の状態とか，それを表す出力関数 control_stateを特別視せず，全ての出力関数を同じように扱っている．本報告では，任意の状態sにおいてVALID(T(s))が真となる状態遷移は高々１つ，すなわち次の状態遷移が（存在するなら）一意に決まるような順序回路を対象とする．

以上の記述スタイルは，抽象的な機能記述レベル（例えばCPUでは一命令実行の意味の記述）から具体的な論理設計レベル（例えばレジスタ，カウンタ等の各入力線に対する論理式の記述）まで一貫して採用される．

ソフトウェアの代数的記述における実現の定義は「上位レベルの記述で合同な表現式対は，下位レベルの記述でも対応する表現式対が合同になっている」というものであるが，VALID関数に関しては，そのような定義では不十分である．なぜならば，この条件だけでは，上の記述で許される状態遷移に対応する（下の記述における）状態遷移が存在するということだけを規定しており，下の記述にはそれ以外の動きが存在してもよいことになる．VALID関数は回路の動きを定めるものであるから，下の記述においては，上で記述された動きだけをちょうど実行するように，下のレベルのVALID関数を記述せねばならない．

以下，まず順序回路と実現の定義を述べ，続いて段階的詳細化について説明する．そして，簡単なCPU記述の一部を例に，様々な抽象化レベルの記述の仕方や，レベル間の対応（実現）について述べる．

2．順序回路の定義

代数的言語ASL/*では[5]，従来の“項”に相当する“表現式”の集合を，文脈自由文法で指定し，表現式集合上の合同関係を公理で定義する．ASL/*には，既に定義されている合同関係に於て合同な表現式対を公理として包含する“射影”という機能がある．この機能により，記述の前提となる基本的なデータタイプや関数などの定義と，記述対象の構造や性質の定義とを分離することができる．

基本関数には整数やブールに関する通常の演算やIF関数，並び[,,…]等を含む．メモリは「抽象的テーブル」と考え，GET,PUTを基本関数として扱うこともある．

［定義１］（順序回路の定義）
代数的記述が次の形をしているとき，その代数的記述を順序回路と呼ぶ．

（１）前提とする基本関数（基本定数などを含む）以外の定数及び関数は次のように分類される．
- (a) 初期状態を表す定数（１個のみ．INITまたはinitで表わす）
- (b) 状態遷移関数（複数．Tまたはt，及びそれらに添字をつけた記号で表わす）
- (c) 出力関数（複数．Fまたはf，及びそれらに添字をつけた記号で表わす）
- (d) VALID関数（１個のみ．VALIDまたはvalidで表わす）

（２）(a)～(e)の関数に関する公理は次の形のもののみからなる．

出力関数：
- (イ) F(INIT)==C
- (ロ) F(T(s))==E(s)

VALID関数：
- (ハ) VALID(INIT)==TRUE
- (ニ) VALID(T(s))==VALID(s) and B(s)

但しCは基本関数からなる定数式．E(s)，B(s)は，状態変数s，出力関数及び基本関数のみからなる表現式

状態以外の引数を持つ状態遷移関数や，状態の対応で一つの状態が複数の（部分）状態の並びに対応するような対応関係も考えられるが，ここでは簡単のため上記のような形で議論する．

(イ)の公理は，初期状態における各出力の値を与える．(ロ)は，各状態遷移における各出力の値の変化を与える．

VALID(s)は，状態sが初期状態以降許される状態遷移のみから得られた状態であるか否かを表わしていると解釈する．よって(ハ)は，初期状態は常に許される状態であることを表わしている．VALID(T(s))という形の記述に着目するならば，これは状態sから遷移Tを行うべき条件を表していると解釈できる．(ニ)は，各状態における許される状態遷移を規定し，状態遷移の実行順を与えている．また，VALID関数の値がある状態において偽になれば，それ以降の状態ではVALID関数の値は常に偽であることを表わしている．以下，任意の状態sにおいて，VALID(T(s))が真となるような状態遷移は高々1つであるとする．

順序回路の状態は，初期状態INITに何回かの状態遷移関数を施した表現式であるが，便宜上，状態を，初期状態以降その状態に至るまで適用された状態遷移関数の系列で表す．例えば

T2(T1(INIT))

は

INIT・T1・T2

と表す．また，VALID(s)≡TRUEとなるような状態をVALIDな状態とよぶ．

ISを状態を引数とする述語とする．状態αの初期部分列を順にα_1(=INIT)，α_2，…，α_n(=α)とする（即ち，初期状態からαに到達するまでに経由した状態を順にα_1，α_2，…，α_nとする）．状態αの初期部分列のうち，IS(α_i)が真となるような各状態α_iにおける全出力関数値の組＜F1(α_i)，F2(α_i)，…＞の系列をαの出力履歴と呼び，$O_{IS}(\alpha)$と書く．

3．順序回路の実現

同じ基本関数を前提とする2つの順序回路A，Bを考える．

spec A	spec B
$F_i(INIT)$==…	$f_i(init)$==…
$F_i(T_j(s))$==…	$f_i(t_j(s))$==…
…	…
VALID(INIT)==…	valid(init)==…
$VALID(T_j(s))$==…	$valid(t_j(s))$==…
…	…

Aの各出力関数Fiに対し，$EXP_{Fi}(s)$をBの出力関数及び基本関数及び変数sのみからなる表現式とする．この時

$F_1(s)$==$EXP_{F1}(s)$

$F_2(s)$==$EXP_{F2}(s)$

…

をBからAへの出力の対応と呼ぶ．EXP=＜EXP_{F1}，EXP_{F2}，…＞とおくとき，BからAへの出力の対応を単にEXPで表すこともある．Aの出力履歴$O_{IS}(\alpha)$とBの出力履歴$O_{is}(\beta)$において，両者の値が等しく，出力履歴の中で対応する出力がEXPのもとで等しいとき，$O_{IS}(\alpha)$と，出力履歴$O_{is}(\beta)$は出力の対応EXPのもとで等価と言う．

［定義2］

2つの順序回路AおよびB（Bはいわゆる無矛盾とする）と，BからAへの出力の対応EXP，Aの状態に関する述語IS，Bの状態に関する述語isが与えられているものとする．そのとき，下記条件（1）および（2）が成り立つ時，かつその時のみ，EXP，IS，isのもとでBはAの実現であるという．

（1）Aの全てのVALIDな状態αに対して，BのVALIDな状態βが存在し，αの出力履歴$O_{IS}(\alpha)$とβの出力履歴$O_{is}(\beta)$はEXPのもとで等価である．

（2）Bの全てのVALIDな状態βに対して，次の条件を満たすAのVALIDな状態αとBのVALIDな状態β'が存在する．$\beta'=\beta\cdot\gamma$と表すことができ（但しγは，Bの状態遷移関数の系列で空系列を含む），αの出力履歴$O_{IS}(\alpha)$とβ'の出力履歴$O_{is}(\beta')$はEXPのもとで等価である．

上記条件の（1）より，Aの動きに対応するBの動きが存在し，逆に（2）から，Bの動きに対応するAの動きも存在する．VALIDな状態遷移が，各状態において高々1つという条件のもとでは，AのISのもとでの出力列（一意に決まる）とBのisのもとでの出力列（一意に決まる）とは完全に一致する．

4．順序回路の詳細化

IC（例えばカウンタやレジスタ）や，あるいは，それらで構成される基本回路等は，状態遷移関数と出力関数を定義したモジュールとみることができる．そこで順序回路を構成する関数及び公理のうち状態遷移と出力関数に関するもののみからなるものをモジュールと呼ぶ．モジュールは，一般に抽象度の高いレベルでは，書き手が適当に定め，具体的なレベルでは，既存のIC等から必要なものを適当に選ぶことによって決められる．用いられるモジュールが決まっているとして，そのモジュールを用いた順序回路の詳細化の問題を以下のように定式化する．

順序回路AとAの状態に関する述語IS，及びモジュールMが与えられたとき，BからAへの出力の対応とBの状態に関する述語isを求め，Mを含む順序回路でAの実現であるような順序回路B（出力関数を新たに導入してもよい）を求めることを（Mを用いた）BによるAの詳細化という．

順序回路Bでは，用いるモジュールの状態遷移関数と出力関数は与えられているから，詳細化の時に追加するものは，次の3つである．

（ア）BからAへの出力の対応を与える複合関数とその公理．

（イ）モジュールMの動きを規定するVALID関数validの公理とその記述．

（ウ）Bの状態に関する述語isの公理の記述

与えられたA，Mに対して，上記の（ア）～（ウ）を求めるために，次のような方法が考えられる．

（1）B（実質はM）からAへの出力の対応 EXP = $<EXP_{F1}, EXP_{F2}, \cdots>$ を考案する．

（2）ソフトウェアの詳細化で普通行われるように，Aの各状態遷移関数T_iとBの状態遷移との対応を表す表現式

$T1(s)==COR_{T1}(s)$

$T2(s)==COR_{T2}(s)$

…

($COR_{Ti}(s)$は，基本関数，変数s，およびBの状態遷移関数と出力関数からなる表現式．初期状態の対応INIT==initを含む）を考案する（これをBからAへの遷移の対応と呼ぶ）．全ての対応の組をCOR=$\langle COR_{T1}, COR_{T2}, \cdots\rangle$とおくとき，BからAへの遷移の対応を単にCORと書く．遷移の対応CORの公理を用いてAの状態αを書き換えて得られるBの状態βを$TS_{COR}(\alpha)$(Transferred State）と書く．

（3）構造的帰納法などを用いて，Aの全てのVALIDな状態α及び各出力関数F_iについて出力の対応と遷移の対応が“正しい”こと，すなわち

$F_i(\alpha) \equiv_A C \Leftrightarrow EXP_{Fi}(TS_{COR}(\alpha)) \equiv_M C$

が成り立つことを証明する．上式が成立するための十分条件は，AのVALID関数以外の公理

$F(\cdots) == r$

に対し，Mの公理，出力の対応及び状態の対応からなる公理系において，定理

$F(\cdots) \approx r$

が成り立つことである．この証明は，代数的仕様検証支援系[7]などを用いて行う．

（4）Aの状態遷移に対してBが遷移の対応で指定された状態遷移系列を忠実に実行するように，BのVALID関数validの公理を導く．

そのためには，状態sが，COR_{Ti}のどこまで実行した状態かを表す出力関数CONTROL(s)を新たに（Bに）導入する．CONTROLの値は次のように決める，$COR_{T1}, COR_{T2}, \cdots$に現れる全ての状態遷移関数に識別子を割り当て（同じ関数が複数回現れた場合も異なる識別子を割り当てる），状態遷移後，その状態遷移に割り当てられた識別子をCONTROL関数の値とする．

このようなCONTROL関数があれば，COR_{Ti}中に現れる一つの状態遷移tについて，そのtが実行される必要十分条件は，tが最初に実行される状態遷移なら，T_iの条件（$VALID(T_i(s))$）に等しく，そうでなければ，CONTROL関数値が直前の状態遷移の識別子に等しいときである．tは一般に複数回現れるので，VALID(t(s))を，それら全てのtの出現について，tが実行される条件の論理和とする．

（5）次のようなisの公理を導出する．

$IS(\alpha) \equiv$ TRUEなる（Aの）全ての状態αについてis$(TSCOR(\alpha)) \equiv$TRUE，それ以外のBの状態βについてis$(\beta) \equiv$FALSE

このことを表現するisの公理の右辺は，与えられているISの公理の右辺を出力の対応に従って書き換えて得られる表現式と，Aの一つの状態遷移に対応するBの一連の状態遷移が完了したか否かの述語pとの論理積とすればよい．

上記（4）のvalidの公理の作り方により，以下が成立する．

（a）Aの全てのVALID状態αに対して$TS_{COR}(\alpha)$はVALIDである．

（b）Bの全てのVALID状態βに対して，

$TS_{COR}(\alpha) = \beta \cdot \gamma$

なるBの状態遷移関数の系列γおよびAのVALID状態αが存在する．

従ってBが無矛盾であること，（3）が証明されれば，Aの全てのVALIDな状態αに対して，出力履歴$O_{IS}(\alpha)$と$O_{is}(TS_{COR}(\alpha))$とがEXPのもとで等価になるからBはAの実現になっている．

順序回路AとBの抽象度の差が大きい場合，上記（3）の証明，及び（4）のBのVALID関数の導出は，一般に複雑になる．そのような場合，順序回路Aをモジュールを用いて順序回路Bとして実現し，さらに順序回路Bをモジュールより簡単な（またはより具体的な）モジュールNを用いて順序回路Cとして実現

するというように，段階的に詳細化していけばよい．

［例１］抽象レベルの記述

（１）記述対象

プログラムカウンタ(PC)，アキュムレータ(ACC)，メモリ(MEM)等から構成される簡単なマイクロコンピュータを考える．ここでは簡単のため，メモリアクセスの仕方などにより，命令をマシン制御命令，分岐命令，LOAD命令，STORE命令，演算命令の５グループに分類し，各グループから代表的な命令を記述の対象として選ぶことにする．マシン制御命令は，アドレス部を持たない命令グループで，NOP（停止命令），SHL（ACCのシフト命令）等がある．分岐命令は，制御を移すもので，ジャンプ命令，条件付きジャンプ命令がある．LOAD，STORE命令は，MEMとACC間のデータの転送に用いられ，演算命令は，ACCとMEM中のデータとの演算結果をACCに格納するタイプの命令で，四則演算，論理演算等がある．各グループから以下の命令を選んだ．

マシン制御命令：HLT（停止）
　　SHL（ACCの内容を１ビット，左シフトする）
分岐命令　：BRM（ACCの内容が負の時，アドレスの指す先の命令に制御を移す）
LOAD命令　：LDA（アドレスの指すメモリ番地のデータをACCにロードする）
STORE命令　：STA（ACCの内容をアドレスの指すメモリ番地に格納する）
演算命令　：ADD（ACCの内容とアドレスの指すメモリ番地のデータを数と見なして和をとり，結果をACCに入れる）

データ長は，８ビットを１語とし，命令語及びデータは１語，アドレスは２語（第１語が下位アドレス，第２語が上位アドレス）で表すものとする（語長や一命令を構成する語数なども抽象的に書けるが，ここでは簡単のため具体的数値を用いる）．MEM及びPCの初期値はパラメータ化し，任意に与えられるようにする．アドレス方式は，簡単のため，直接（命令部のすぐ次の２語がオペランドまたは分岐先アドレスと解釈し処理する）方式のみとする．HLT，SHLは命令部のみの１語命令，他は命令語とアドレス部からなる３語命令である．

（２）命令の意味定義

上記（１）で述べたマイクロコンピュータの各命令の意味は，以下のように記述することができる．マイクロコンピュータの状態は抽象状態とし，PC,ACC,MEM等は抽象状態からそれぞれの値を取り出す同名の出力関数で表すことにする．また，HLT命令を実行したか否かを表すSTOPというフラグ（出力関数で表す）を用意する．１命令の実行を状態遷移関数CYCLEで表し，各出力関数が状態遷移CYCLEに実行によりどのように値が変わるかを公理で記述する．

出力関数の公理の書き方についてPCを例に用いて説明する．PCは，初期値init_pcが与えられる．１命令実行すると，実行する命令がBRM命令で，かつACCの内容が負ならば，命令語の次に格納されているアドレス部の値がPCの値となる．それ以外の場合はPCの値は実行する命令の命令長に応じて増やされる．このことは，次のように表される（本報告では，簡単のため構文指定のための文法や，公理の変数のタイプ指定の記述は省略する）．

```
PC(INITIAL)  == init_pc;
PC(CYCLE(s)) ==
    if OP_TYPE(s)=BRM and ACC(s)<0
    then
        OPERAND_ADR(s)
    else
        PC(s)+(if OP_TYPE(s)=HLT or
                        OP_TYPE(s)=SHL
               then 1
               else 3);
```

OP_TYPE()及びOPERAND_ADR()は，それぞれ現在実行中の命令の命令コード及びアドレス部を表す補助関数で，以下のように定義される．

```
OPERAND_ADR(s) ==
      APPEND(GET(MEM(s),PC(s)+2),
                  GET(MEM(s),PC(s)+1))
OP_TYPE(s)     == OP(GET(MEM(s),PC(s)))
```

ここで，GETは抽象的な“表”から値を取り出す基本演算，APPENDは，２つのビット列を連結して整数を構成する基本演算である．またOPは，現段階では値が定義されていない関数（具体的レベルで意味が定義される）である．

VALID関数は，STOPの値がLの間状態遷移CYCLEを実行するように定義すればよく，次のようになる．

```
VALID(INITIAL)  == TRUE;
VALID(CYCLE(s)) == VALID(s) and STOP(s)=L;
```

PC,VALID以外の出力関数の公理の記述を付録１に記載する．

［例２］　詳細化の例

例１で述べたマイクロコンピュータを用いて，詳細化の仕方について説明する．

（１）モジュールの決定

CPUの動作原理について述べている教科書等には，命令の実行をいくつかのフェーズに分け，各フェーズでの動作の説明がよく書かれている．ここでは，メモリへのアクセスを，メモリアドレスレジスタ（MAR，２バイト）へアドレスを書き込んでから行うこと，及び命令コードの解析を，一旦MEMから命令レジスタ（IR，１バイト）に命令語を移してから行うことにし，各命令の実行を，次の（最大）４つの動作（ADSET1，FETCH，ADSET2，EXEC）に分ける．

ADSET1：命令語をMEMから読み出すために，PCの内容をMARに移す．

FETCH：MEMから命令語をIRに移す．また，PCは次の命令語または命令のアドレス部を指すように１増やす．

ADSET2：命令のアドレス部をMARに入れ，PCの値を２増やす．STA，LDA，ADD命令時はオペランドのアドレスが，BRM命令時は，分岐アドレスがMARに入ることになる．

EXEC：命令語に対応する操作を行う．

PCを例に各状態遷移関数後の出力関数値を記述する．（１）で述べた，各状態遷移関数の意味から，PCの値は，ADSET1後は値を変えず，FETCH後は，１増やし，ADSET2後は，２増やす．EXECを実行したときは，BRM命令で条件が成り立ったときMARの値がセットされ，条件が成り立たなかった時またはその他の命令では，値は変わらない．このことは次のように表される．

```
PC(INITIAL)   == init_pc;
PC(ADSET1(s)) == PC(s);
PC(FETCH(s))  == PC(s)+1;
PC(ADSET2(s)) == PC(s)+2;
PC(EXEC(s))   ==
    if OP(IR(s))=BRM and ACC(s)<0 then
        MAR(s)
    else PC(s);
```

PC以外の出力関数の公理を付録２に記載する．付録２では，状態遷移後の値が何であってもよい出力関数については定義していない．例えばIRについては，命令実行後，次にMEMから命令をIRに読み込むまで値が何であっても良いので，EXEC後及びADSET1後の値を定義していない．

（２）出力と遷移の対応

このレベルの出力関数は，例１のレベルで用いた同名の出力関数に対応している．状態遷移関数CYCLEの実行は，次のような状態遷移の系列に対応する．

```
s•CYCLE ==
    s•ADSET1•FETCH•if OP(IR)=HLT or OP(IR)=SHL
                   then
                       EXEC
                   else
                       ADSET2•EXEC
```

この対応のもとで，４.の（３）の証明を行う．

（３）VALID関数の記述

４つの状態遷移がどのような順序で行われるかを定めるVALID関数の記述を説明する．４つの状態遷移のうち，ADSET2は命令にアドレス部を持たないHLT，SHL命令に対しては実行されない．HLT，SHL命令はADSET1，FETCH，EXECの順に，その他の命令はADSET1，FETCH，ADSET2，EXECの順に行う．このため現在どのフェーズを実行したのかを保持しておくことにし，その情報を（上記のCONTROL関数に対応する）出力関数PHASEで表す．このようにすれば，例えばADSET2は，PHASEの値がfetchで，命令コードがADD，STA，LDA，BRMの時実行を行うように，PHASE，VALID関数を次のように記述すれば良い．

```
PHASE(INITIAL)   == start;
PHASE(ADSET1(s)) == adset1;
PHASE(FETCH(s))  == fetch;
PHASE(ADSET2(s)) == adset2;
PHASE(EXEC(s))   == start;

VALID(INITIAL)   == TRUE;
VALID(ADSET1(s)) == VALID(s) and
    PHASE(s)=start and STOP(s)=L;
VALID(FETCH(s))  == VALID(s) and
    PHASE(s)=adset1;
VALID(ADSET2(s)) == VALID(s) and
    PHASE(s)=fetch and
      (OP(IR(s))=ADD or OP(IR(s))=STA or
       OP(IR(s))=LDA or OP(IR(s))=BRM);
VALID(EXEC(s))   == VALID(s) and
     PHASE(s)=fetch and
       (OP(IR(s))=HLT or OP(IR(s))=SHL)
 or  PHASE(s)=adset2;
```

初期状態において，PHASEの値がstartかつSTOPの値がLなので，VALID(ADSET1(s))がTRUEとなる．従って状態遷移ADSET1が実行され，PHASEはadset1となる．すると次にVALID(FETCH(s))がTRUEとなり，FETCHが実行される．このとき命令語がIRにセットされ，PHASEはfetchとなる．次は命令語がSTA，LDA，ADD，BRM命令の時はVALID(ADSET2(s))が，それ以外の時はVALID(EXEC(s))がTRUEになる．例えば，命令がADDの時はADSET2 が実

行され，PHASE にadset2がセットされる．次に VALID (EXEC(s))がTRUEになり，EXEC実行後，PHASE の値もstartにもどり再び次の命令実行が行われる．また，命令がHLTの時は，ADSET1,FETCH実行後,VALID(EXEC(s))がTRUEになり，EXECが実行される．その結果STOPがHになりPHASEはstartに戻る．しかしその後は，VALID関数は全ての状態遷移関数に対してFALSEの値をとる．すなわちどの状態遷移も行ってはいけないことを表している．

例１のレベルの全状態についてIS関数を真としたとき，このレベルではEXEC実行後についてのみIS関数を真とすれば，このレベルの記述が，例１のレベルの記述の実現になっている．

5．具体的レベルの記述について

ここでは，具体的な部品により順序回路を構成したときの各部品の入力論理を，上述の枠組みの中で記述する方法を説明する．

［例３］部品の抽象的論理入力の記述例

（１）部品の記述例

回路設計では，レジスタ，カウンタなど既存のMSI，LSIを部品として用いることが多い．その場合，各部品はデータの入出力線と制御入力線を持ち，クロック入力毎に内部状態を変える１つの機能素子として扱われる．

カウンタ（PC,MARに16ビット，PHCに3ビットのものを使用）を例に，部品の記述例を説明する．状態遷移関数は，クロックに相当するCK$_{CNT}$のみとし，その第１引数にカウンタの抽象状態が，第２引数にカウンタの行う機能が，第３引数はカウンタにロードするときの入力データが与えられるとする．カウンタの出力データ（カウンタが保持している値）は，出力関数OUT$_{CNT}$で表す．これらの関数の意味は次のように定義することができる．（mod Mの Mは 2^{16}等の整数）

```
OUTCNT(CKCNT(s, HOLD,d)) == OUTCNT(s);
OUTCNT(CKCNT(s,   UP,d)) == OUTCNT(s)+1 mod M;
OUTCNT(CKCNT(s,CLEAR,d)) == 0;
OUTCNT(CKCNT(s, LOAD,d)) == d;
```

その他，ロード／ホールド機能だけのレジスタ（ADH,ADL,IRに使用），READ/WRITE可能なメモリ，Ｄフリップフロップ(DFF，STOPbit保持に使用），演算機能付きシフトレジスタ（ACCに使用）を部品として使用する．これらの定義は省略する．

（２）部品の入力論理の記述例

マイクロコンピュータの抽象状態を，部品の抽象状態の並び［phs,stops,pcs,accs,mems,irs,mars,adhs,adls］で表す．各部品は左から,PHC(例２のPHASEに対応するカウンタ),STOP,PC,ACC,MEM,IR,MAR，ADH,ADLに対応している．部品の入力論理はVALID関数の公理（付録３）の中で記述する．付録３では，公理の左辺は次のようになっている．

```
VALID( [CKCNT(phs,phc_ctl,phc_in),
        CKDFF(stops,stop_ctl),
        CKCNT( pcs, pc_ctl, adr_bus),
        CKALU(accs,acc_ctl,data_bus),
        CKMEM(mems,mem_ctl,data_bus,mem_adr),
        CKREG( irs, ir_ctl,data_bus),
        CKCNT(mars,mar_ctl, adr_bus),
        CKREG(adhs,adh_ctl,data_bus),
        CKREG(adls,adl_ctl,data_bus)] ) ==
```

各部品の抽象状態である第１引数（phs,stops等）を除く他の変数（すなわち各部品の入力線）について，そこへの入力をどのような値にして各部品の状態遷移を行えばVALIDが真であるかをVALID関数の右辺で記述する．付録３でVALID関数の右辺は，

部品の入力変数　＝　式

の形の項の論理積の形で書かれている．例えば，PCの抽象的な制御信号入力を表すpc_ctlに関する項は次の通りである．

```
pc_ctl =
    if OUTCNT(phs)=7 and OP(irs)=BRM
       and OUTALU(accs)<0 then
         LOAD
    else if OUTCNT(phs)=1
         or OUTCNT(phs)=3 or
            OUTCNT(phs)=5
    then UP
    else HOLD
```

各部品の入力は，VALID関数がTRUEとなるように決めなければならない．そのためには，pc_ctlに関して上記の項がTRUEになるように（抽象的な）配線を行うことになる．すなわち，上記等式の右辺を表す（この段階では抽象的な）組合せ回路の出力をpc_ctlへ入力すればよい．

（３）バスの表現方法

多くのレジスタ類が相互にデータをやり取りするとき，回路を簡単にするため，バスを用いることがある．バスは，各レジスタの入力が一定のビット幅を持った一組のデータ線を共有し，一時に１つのレジスタのみがそのデータ線への出力を許されるというものである．入力が一本のデータ線を共有することは，上に述べたVALID関数の公理の左辺中に，同じ変数が複数個現れる

ことで表される．この例では，data_busとadr_busの２本のバスがある．バスへどのレジスタの出力が流れるかはVALIDの公理の右辺で記述する．data_busについては，

```
data_bus = if mem_ctl=READ then
             OUTMEM(mems,mem_adr)
           else
             OUTALU(accs)
```

の様に記述されており（mem_ctlがREADに等しくするための条件は別に書かれている．付録３参照），この場合メモリかアキュムレータのいずれか一方が出力される．adr_busについても，同様にPCの出力，または，〈ADH，ADL〉のいずれか一方が出力される．

［例４］具体的入力の記述例

ここでは，付録３の抽象的な入力論理の記述から，実際の部品の入力線の論理式を導出する方法を，PCの制御信号入力を例として説明する．例３ではPCの制御信号入力を，抽象的な値｛HOLD,UP,CLEAR,LOAD｝を用いて記述した．これらの値を次のように符号化する．

	〈pc_ctl1 , pc_ctl2〉
HOLD	〈 FALSE , FALSE 〉
UP	〈 TRUE , TRUE 〉
CLEAR	〈 TRUE , FALSE 〉
LOAD	〈 FALSE , TRUE 〉

ここでTRUE,FALSEは例えば電圧の高低など，実際の回路で定義されている値である．pc_ct12がTRUEになるのは上記の表よりUPとLOADの場合であるから，pc_ctlの式より，その論理式は次のようになる．

```
pc_ct12 = (OUTCNT(phs)=7 and
           OUTREG(irs)=BRM and
           OUTALU(accs)〈0) or
           OUTCNT(phs)=1 or
           OUTCNT(phs)=3 or
           OUTCNT(phs)=5
```

上式で，OUTCNT(phs)=n(n=1,3,5,7)及びOUTREG(irs)=BRMについてはPHC，IRの出力にデコーダを取り付け，それぞれ対応する（デコーダの）出力線の値を用いればよく，OUTALU(accs)〈0は，ACCの出力のうち符号を表す最上位ビットの出力線の値を用いればよい．pc_ctl1についても同様にして次のような論理式が得られ，その他の部品の入力線についても同様に記述することができるが，ここでは省略する．

```
pc_ctl1 = OUTCNT(phs)=1 or
          OUTCNT(phs)=3 or
          OUTCNT(phs)=5
```

ここでの記述は，例３の多値入力線を複数の２値入力線で表し，各入力線の値を記述しているに過ぎないので，例３と同じ形式で記述することができる．

６．あとがき

以上，代数的言語ＡＳＬを用いた順序回路の記述及び実現の定義について述べた．また，詳細化するときの１つの標準的と思われる方法を提案した．筆者らのグループでは，支援系により詳細化のプロセスの一部を自動化することを検討しており，すでに，一定の書式のもとで，下位のVALID関数の自動生成を行うアルゴリズムを実現している．

参考文献

[1] G.C.Gopalakrishnan,M.K.Srivas and D.R.Smith："From Algebraic Specification to Correct VLSI Circuits",in D.Borrione(ed.):"from HDL descriptions to guaranteed correct circuit designs",North-Holland,pp.197-225(1987).

[2] G.C.Gopalakrishnan,D.R.Smith and M.K.Srivas："An Algebraic Approach to the Specification and Realization of VLSI designs",in C.J.Koomen and T.Moto-oka(eds.):"Computer Hardware Description Languages and their Applications",North-Holland,pp.16-38(1985).

[3] S.dasgupta and J.Heinanen："on the Axiomatic Specification of Computer Architectures",in C.J.Koomen and T.Moto-oka (eds.):"Computer Hardware Description Languages and their Applications", North-Holland,pp.1-15(1985).

[4] 嵩，谷口，杉山："代数的言語の設計と処理系"，榎本編："ソフトウェア工学ハンドブック"，オーム社(1986-06).

[5] 嵩，谷口，杉山，関："代数的言語ＡＳＬ／＊－意味定義を中心に－"，信学論（Ｄ），J69-D,7, pp.1066-1074(1986-07).

[6] 東野，工藤，縄田，杉山，谷口："代数的仕様検証系及びそれを用いた検証例"，信学論（Ｄ），J67-D,4, pp.472-479(1984-04).

[7] 横山，谷口：“ハードウェアの代数的仕様記述とその具体化について”，
信学技報，Vol.87,No.181,COMP87-34,(1987-09).

付録1　例1のレベルのアキュムレータ，メモリ，ストップビットの記述

```
ACC(CYCLE(s)) ==
    if OP_TYPE(s)=SHL then
        shift_left(ACC(s))
    else if OP_TYPE(s)=LDA then
        GET(MEM(s),OPERAND_ADR(s))
    else if OP_TYPE(s)=ADD then
        ACC(s) + GET(MEM(s),OPERAND_ADR(s))
    else ACC(s);

MEM(INITIAL)  == init_mem;
MEM(CYCLE(s)) ==
    if OP_TYPE(s)=STA then
        PUT(MEM(s),OPERAND_ADR(s),ACC(s))
    else MEM(s);

STOP(INITIAL)  == L;
STOP(CYCLE(s)) == if OP_TYPE(s)=HLT then H
                  else STOP(s);
```

付録2　例3のレベルの出力関数の記述

```
PC(INITIAL)   == init_pc;
PC(ADSET1(s)) == PC(s);
PC(FETCH(s))  == PC(s)+1;
PC(ADSET2(s)) == PC(s)+2;
PC(EXEC(s))   ==
    if OP(IR(s))=BRM and ACC(s)<0 then
        MAR(s);
    else PC(s);

MEM(INITIAL)   == init_mem;
MEM(ADSET1(s)) == MEM(s);
MEM(FETCH(s))  == MEM(s);
MEM(ADSET2(s)) == MEM(s);
MEM(EXEC(s))   ==
    if OP(IR(s))=STA then
        PUT(MEM(s),MAR(s),ACC(s))
    else MEM(s);

STOP(INITIAL)   == L;
STOP(ADSET1(s)) == STOP(s);
STOP(FETCH(s))  == STOP(s);
STOP(ADSET2(s)) == STOP(s);
STOP(EXEC(s))   ==
    if OP(IR(s))=HLT then H
    else STOP(s);

ACC(ADSET1(s)) == ACC(s);
ACC(FETCH(s))  == ACC(s);
ACC(ADSET2(s)) == ACC(s);
ACC(EXEC(s))   ==
    if OP(IR(s))=SHL then
        shift_left(ACC(s))
    else if OP(IR(s))=LDA then
        GET(MEM(s),MAR(s))
    else if OP(IR(s))=ADD then
        ACC(s) + GET(MEM(s),MAR(s))
    else ACC(s);

IR(FETCH(s))  == GET(MEM(s),MAR(s));
IR(ADSET2(s)) == IR(s);

MAR(ADSET1(s)) == PC(s);
MAR(FETCH(s))  == MAR(s);
MAR(ADSET2(s)) == OPERAND_ADR(s);

PHASE(INITIAL)   == start;
PHASE(ADSET1(s)) == adset1;
PHASE(FETCH(s))  == fetch;
PHASE(ADSET2(s)) == adset2;
PHASE(EXEC(s))   == start;

VALID(INITIAL)   == TRUE;
VALID(ADSET1(s)) == VALID(s) and
    PHASE(s)=start and STOP(s)=L;
VALID(FETCH(s))  == VALID(s) and
    PHASE(s)=adset1;
VALID(ADSET2(s)) == VALID(s) and
    PHASE(s)=fetch and (OP(IR(s))=ADD or
    OP(IR(s))=STA or OP(IR(s))=LDA or OP(IR(s))=BRM);
VALID(EXEC(s))   == VALID(s) and
    PHASE(s)=fetch and
    (OP(IR(s))=HLT or OP(IR(s))=SHL) or
    PHASE(s)=adset2;
```

付録3　部品の入力論理の記述

```
VALID( [CKCNT(phs,phc_ctl,phc_in),
        CKDFF(stops,stop_ctl),
        CKCNT( pcs, pc_ctl, adr_bus),
        CKALU(accs,acc_ctl,data_bus),
        CKMEM(mems,mem_ctl,data_bus,mem_adr),
        CKREG( irs, ir_ctl,data_bus),
        CKCNT(mars,mar_ctl, adr_bus),
        CKREG(adhs,adh_ctl,data_bus),
        CKREG(adls,adl_ctl,data_bus)] ) ==
    OUTDFF(stops) = L
    and
      mem_adr  = OUTCNT(mars)
    and
      data_bus = if mem_ctl=READ then
                    OUTMEM(mems,mem_adr)
                 else
                    OUTALU(accs)
    and
      adr_bus  = if pc_gate=OPEN then
                    OUTCNT(pcs)
                 else
                    <OUTREG(adhs),OUTREG(adls)>
    and
      phc_ctl  =
          if OUTCNT(phs)=2 and
            (OP(irs)=HLT or OP(irs)=LSH) or
             OUTCNT(phs)=7 then
               CLEAR
          else UP
    and
      phc_in  = 0
    and
      stop_ctl =
          if OP(irs)=HLT then
               SET
          else HOLD
    and
```

```
  pc_ctl =
      if OUTCNT(phs)=7 and OP(irs)=BRM and
         OUTALU(accs)<0
      then LOAD
      else if OUTCNT(phs)=1 or OUTCNT(phs)=3 or
              OUTCNT(phs)=5
      then UP
      else HOLD
and
  pc_gate  =
      if OUTCNT(phs)=0 then
           OPEN
      else CLOSE
and
  acc_ctl  =
      if OP(irs)=LSH and OUTCNT(phs)=2 then
           LSH
      else if OP(irs)=ADD and OUTCNT(phs)=7 then
           ADD
      else if OP(irs)=LDA and OUTCNT(phs)=7 then
           LOAD
      else HOLD
and
  mem_ctl  =
      if  OP(irs)=STA and OUTCNT(phs)=7 then
           WRITE
      else if OUTCNT(phs)=1 or OUTCNT(phs)=3 or
              OUTCNT(phs)=5 or OUTCNT(phs)=7 and
              (OP(irs)=LDA or OP(irs)=ADD) then
           READ
      else HOLD
and
  ir_ctl  =
      if  OUTCNT(phs)=1 then
           LOAD
      else HOLD
and
  mar_ctl =
      if OUTCNT(phs)=0 then
           LOAD
      else if OUTCNT(phs)=2 or OUTCNT(phs)=4 then
           UP
      else HOLD
and
  adh_ctl =
      if OUTCNT(phs)=5 then
           LOAD
      else HOLD
and
  adl_ctl =
      if OUTCNT(phs)=3 then
           LOAD
      else HOLD;
```