設 計 自 動 化 46－11
（１９８９．２．２１）

ハードウェア記述言語
UHDL

藤沢　久典　　藤田　昌宏　　川戸　信明
（株）富士通研究所

新しいハードウェア記述言語であるUHDL(Unified Hardware Description Language)を開発した．

UHDLは従来のハードウェア記述言語で問題となった点を克服するために次のような特徴を備えた言語である．まず動作記述においてインターバルと言う概念を取り入れることにより同期回路と非同期回路とを区別することなく記述することが可能である．また動作記述は時相論理に基づいているため数学的に厳密に定義でき，検証が容易である．さらに動作記述から論理回路の合成を考えた場合に，現在の技術では設計者が満足できるような回路を合成することは困難である．UHDLでは設計者自らデータパスや構造を指定することにができる．そのためこの情報を利用することによりより品質のよい回路を合成することが可能となる．

ここでは，ハードウェア記述言語UHDLについて，動作記述を中心に説明する．

UHDL : Unified Hardware Description Language

Hisanori FUJISAWA, Masahiro FUJITA, Nobuaki KAWATO

Fujitsu Laboratories Limited

1015 Kamikodanaka, Nakahara-ku, Kawasaki, 211 Japan

We have developed a new hardware description language UHDL. UHDL has the following features to overcome the problem in usual hardware description language.

In the description of behavior, the concept of an interval is introduced. When intervals are used, behavior can be described without distinction of synchronous and asynchronous behavior. Since the description of behavior is based on interval temporal logic, we can treat description of behavior strictly in mathematics and formal verification is easy.

It is difficult to get a high-quality result by automatic circuits synthesis from the description of behavior in the state of the art. In UHDL, the designer can designate a hardware structure by oneself and a synthesis system can synthesize high-quality hardware.

In this paper, we describe UHDL with these features.

1. はじめに

近年における回路の大規模化・複雑化は人手による回路合成を困難なものとし多大な労力と時間を必要とするようになってきた．反面コストの削減や製作行程の短縮は企業においては死活にかかわる重要な課題となっている．このような状況のなか，回路自動合成に対する要求はますます強いものとなってきている．

従来自動化といえばRTL(Register Transfer Level)より下位からの合成が主であったが，最近では動作記述に基づく研究が盛んに行われている．これにともないハードウェア記述言語の重要性がクローズアップされてきた．すなわち不十分なハードウェア記述言語を用いた場合には回路によっては記述することができないものが存在したり，また記述はしたものの合成が困難といった問題点が生じてしまう．そこでハードウェア記述言語は以下の点を考慮したものである必要がある．

(1) 設計者にとって記述が容易であるか？
(2) 回路の記述能力は十分か？
　（設計者の考えた通りの機能の表現が可能か？）
(3) 検証は容易か？
(4) 合成は可能か？

UHDL(Unified Hardware Description Language)は以上のような点を踏まえて新しく開発したハードウェア記述言語である．

2. UHDL(Unified Hardware Description Language)

UHDLは次のような特徴を備えている．

第一に動作，データパス，インターフェイス，構造といった様々な観点からの記述を一つにまとめて記述することができる．ある観点から眺めて記述した情報はviewと呼ばれる単位で管理される．すなわち 対象とする回路に対して複数のviewが存在することになる．これらのviewはモジュールとして一つにまとめて管理される．例えば回路の動作に関しては動作viewと呼ばれるviewで記述され，データパスに関してはデータパスviewに於て記述されことになる．

動作記述をもちいて設計者の満足のいく回路を合成することは現在の技術においては難しく，人手による修正を加えざるを得ない．通常設計者が回路の動作を記述する際にはある程度回路構成を念頭において行う場合が多い．そこで合成の品質を向上させる方法の一つとして設計者の知識を利用する方法がある．UHDLでは設計者自らデータパスや構造の指定を動作記述と同時に行うことができ，より高品質な回路を合成することが可能となる．

またある動作を記述した場合に，異なるデータパス上で回路を実現したいような場合がある．そのような場合には各データパス毎にデータパスviewの記述を行い，各viewに対して異なる名前を付けることにより一つのモジュール内で管理することができる．

第二に動作記述において状態のかわりにインターバルの概念を導入した．インターバルとは状態の概念に時間を加えたものである．これによりステートマシンの苦手としている非同期動作を同期動作と区別なく記述することができる．また動作は時相論理に基づいて記述されるため，数学的に厳密に定義することができ，形式的検証が容易である．

UHDLの記述構成を図１に示す．現在viewとしては管理view(Manage-view)，インターフェイスview(Interface-view)，動作view(Behavior-view)，データパスview(Datapath-view)，構造view(structure-view)が定義されているが今後UHDL支援ツールが充実してくるに従い追加されることも考えられる．

以下各viewについて順に説明していく．

3. インターフェイスview(Interface-view)

インターフェイスviewではインターフェイスすなわち外部から参照する際に必要な情報が記述される．その情報とは以下のようなものである．

(1) 現モジュール内の記述に於て持ちいられている時間の単位．

(2) 外部入出力端子の名前，ビット幅および信号極性の指定．

記述中で用いられている時間は全てここで指定された単位をとるものとする．また外部入出力端子は外部との信号のやり取りを行う端子で，入力端子，出力端子，および双方向の信号が通過するバスの三つのタイプに分けて記述される．各端子について信号のビット幅，外部信号を正論理として扱うか負論理として扱うかの指定（極性指定）をを行う．そのほか外部端子を通過する信号がパルスであった場合にはそのパルス幅および周期に対して制限を加えることができる．インターフェイスviewの記述例を図２に示す．

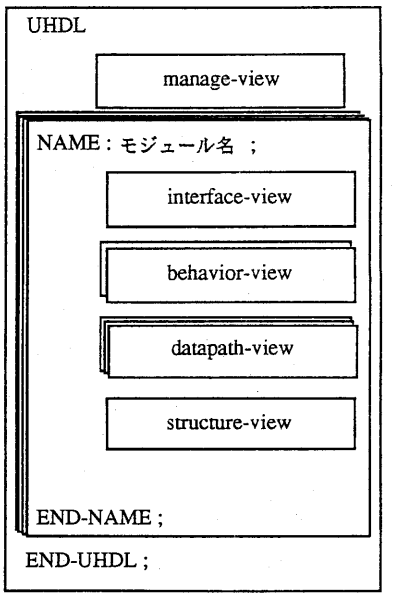


図1　UHDL記述の構成

4. 動作view(Behavior-view)

4.1 時相論理

時相論理とは通常の論理に対して時間の概念を導入したものであり，動作は時相論理に基づいて記述される．ここで用いている時相論理はインターバルという時区間の概念を導入しているためITL（Interval Temporal Logic）と呼ばれる[1, 2]．

インターバルとは始まりと終りの時刻が指定された時区間で．インターバル内の時刻はすべて離散時間として定義される．以下その時刻をインターバル時刻と呼ぶ．状態は各インターバル時刻毎に定義される．今インターバル内にS0から Sn-1 までｎ個の状態があったとするとインターバルの長さはｎ-1で定義される（図3）．時相論理式が真か偽かの判定は指定されたインターバル上で行われる．

S0S1・・Sn-1：時相論理式

上の記述はS0S1・・Sn-1というｎ個の状態を持つインターバル上に於て時相論理式の判定を行うことを意味するものとする．

UHDLにおいて記述される論理式は通常の論理演算子のほか次のような時相演算子を用いて定義される演算子を用いて記述される[2]．

- ○(next)

  S0S1・・Sn-1：○P ≡ S1・・Sn-1：P

- beg

  S0S1・・Sn-1：beg P ≡ S0：P

- fin

  S0S1・・Sn-1：fin P ≡ Sn-1：P

- □(always)

  S0S1・・Sn-1：□P ≡ St・・Sn-1：P
  (0≦∀t≦n-1)

- ；

  S0S1・・Sn-1：P；Q
  ≡ St・・Sn-1：P ∧ St・・Sn-1：Q

- empty

  S0S1・・Sn-1：Emply ⇒ 真(1) n＝1
  偽(0) n＞1

4.2 インターバルとファンクション

UHDLにおけるインターバルは動作記述の基本単位であり次のように記述される．

インターバル名（クロック名）：：実行文・・・；

インターバル時刻はインターバル名の後に指定されたクロック名に対応している．これは同期回路を考えた場合，回路の内部状態はクロックに同期して変化することから考えて自然であろう．

インターバルのうち長さが１のものはDDL[3]等における状態とおなじであり，他のインターバルと区別して次のように記述する．

インターバル名（クロック名）：実行文・・・；

ファンクションは一連の動作が記述されたインターバルをまとめたものでありインターバルは各ファンクションごとに実行される．ファンクションの起動については後述する．

ファンクションが起動されると，LOGICというインターバル名を持つインターバルを除いて最初に記述されたインターバルが実行され，このインターバルが終了すると同時に次に指定されたインターバルが起動される．次に起動するインターバルの指定はGOTO文で行われる．この時前のインターバルの終了時刻と次に起動されるインターバルの最初の時刻はおなじである．もし指定がなければ記述された順に起動される．LOGICインターバルはファンクションが起動している間，ずっと実行状態にある．すなわちファンクションではLOGICインターバル以外のインターバルはインターバルの始まりと終りの時刻を除いて同時に２つのインターバルが実行状態にあることはない．

ファンクションは，LOGICを除く全てのインターバルの実行が終了したかもしくはRETURN 文が実行された場合に終了する．

4.3 インターバルの長さの指定

インターバルの長さはLEN文，DELAY文，STOP文，MORE文の四つの文で指定される（図4）．

LEN文では直接インターバルの長さ（つまりクロック数）を指定する．それに対しDELAY文はクロックに関係なく絶対時間により長さの指定をおこなうものである．
ここでの指定される絶対時間の単位は，インターフェイスviewで定義されたものである．またSTOP文は記述された条件が成立したら現インターバルを終了させる．STOPとDELAYはクロックとは関係のない非同期動作を記述するのに用いられる．MORE文は先の三つと異なりインターバルを延長するかどうかの指定を行うもので，指定した条件が成立したらインターバルは現時刻に於て終了することはなく少なくともあと１時刻以上延長されることを意味する．

以上の記述はすべてインターバル記述の実行文のなかで

```
INTERFACE-VIEW : ikura ;
        PURPOSE : example ;
        REVISION : 009 ;
        DATE : 89/02/01 ;
        DESIGNER : Shimamura ;
        INPUTS : .data, .reset ;
        OUTPUTS : .out ;
        BUSES : .extbus ;
END-VIEW ;
```

図2 インターフェイスviewの記述例

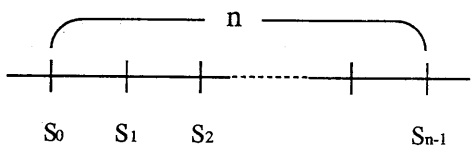


図3 インターバルの状態と長さ

記述される．インターバルの長さに関して複数の指定があった場合にはLEN文，DELAY文，STOP文のうちいづれが一つでも成立すればインターバルは終了するものとし，いづれも成立せずMORE文が成り立った時のみその時刻で終了することはないものとする．また長さの指定がない場合には，システム側でインターバル内の処理がすべて終了したと判断した時点で終了する．

その他インターバルが終了する場合として，CALL文によって起動されたファンクションの強制終了によるものとLOGIC文中におけるNEXT文による終了がある．ファンクションの強制終了については次節にて説明する．NEXT文は次のインターバル時刻で起動すべきインターバルを指定するもので，NEXT文が実行されると次のインターバル時刻で，LOGICインターバル以外の現在起動しているインターバルを強制的に終了させ，指定したインターバルを起動させる．NEXT文は次のように記述される．

NEXT　インターバル名

4.4　ファンクションの起動

ファンクションはCALL文またはEXEC文をインターバル中で実行することによって起動される．以下CALL文またはEXEC文により起動されたファンクションを子ファンクション，CALL文またはEXEC文が記述されたインターバルを親インターバルと呼ぶとする．

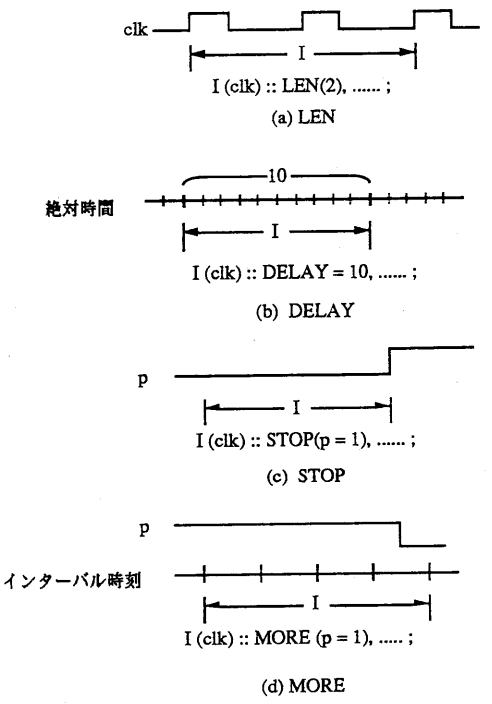


図4　インターバルの指定

CALL文による起動の場合，もし親インターバルの終了に関して指定がない場合には，親インターバルは子ファンクションが終了するのを待ってから終了する．また親ファンクションがLEN文やSTOP文などにより終了した場合には，起動している子ファンクションがあれば状態に関係なくその子ファンクションは強制的に終了させられる．いづれにしても親インターバルは子ファンクションより長く起動状態にあることはあっても先に終了することは決してない．

EXEC文による起動の場合は，ファンクションの起動をおこなうだけであり，親インターバルの終了が子ファンクションに影響を及ぼすことはなく，また逆についても同様である．

4.5　変数

値の参照・代入の対象となるものは大きくレジスタ変数と一時変数の二つのタイプに分類される．

一時変数は回路上のネットに対応し，値を保持する機能は持っていない．そのため値は代入が行われた時刻に於てのみ有効である．レジスタ変数はレジスタやカウンタなど記憶素子に対応し，代入された値は次に代入が行われるまで保持される．ただしレジスタ変数はクロックに同期して動作するため，値の代入は時相代入のみが許される．これに対し一時変数では現在代入のみが許されている．時相代入と現在代入については時節で説明する．

レジスタ変数とハードウェアとの対応はデータパスviewのファシリティ宣言で行われる．一方データパスが指定された場合には，一時変数とデータパスのネットとの対応付けは合成システムが自動的に行う．

4.6　値の代入

値の代入には次の二つの記述がある．一つは現在代入でありもう一つは時相代入である．現在代入は":＝"を用いて記述される．

変数　：＝　論理式

現在代入は論理式のインターバルにおける最初の値を左辺の変数に遅延0で代入を行うものであり，次のように定義される（図5）．

beg（変数）＝beg（論理式）

遅延を指定したいような場合には，

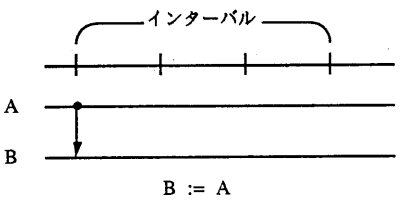


図5　現在代入文

＜変数　：＝　論理式　，　遅延＞

のように記述する．この場合，インターバルにおける最初の時刻の論理式の値を遅延で指定された時間だけ遅れて左辺の変数に代入される．

時相代入は〝＜－〟を用いて次のように表される．

変数　＜－　論理式

時相代入ではインターバルの最初の時刻での論理式の値をインターバルの最後の時刻に左辺の論理式に代入行われる．これを時相演算子を用いて表すと次のようになる（図6）．

fin（変数）＝ beg（論理式）

4.7　並列動作と逐次動作

基本的に各インターバルは逐次的に実行されインターバル内の各記述は並列に実行されるが，非同期動作を記述するような場合には一つのインターバル内で逐次的な実行を記述するしたいような場合がある．そのような場合&&（chop演算子）を用いて記述する．＆&は現在のインターバルを前半と後半に分割するものであり，

A&&B≡A；B

と定義される．図7の記述ではインターバルＩを前半と後半に分けて前半でＰが実行され後半でQが実行される．

これに対し並列動作については「，」出区切ることで表現される．

4.8　演算子

UHDLでは通常の論理式の他，時相論理に基づく演算子を導入している．ここではそれらについて説明する．

@（NEXT演算子）は時相論理をもちいて次のように定義される．

@　≡　○

例えばインターバルが起動して次のインターバル時刻（同期回路では次のクロック）でAの値をBに代入したければ，

@（B：＝A）

のように記述される．また

@（C＜－A）

のような記述も可能である．この場合インターバルが起動してから次のインターバル時刻のAの値がそのインターバルの最後の時刻に於てCに代入される（図8.b）．

FINは時相論理のfinと同じでインターバルの最後の時刻における実行を表す（図9.a）．例えば

FIN（B：＝A）

では図9.bに示すように最後のインターバル時刻でAの値がBに代入される．

KEEPは時相論理を用いて次のように定義される．

KEEP（A）≡　□（¬empty　→A）

KEEPは最後の時刻を除く全ての時刻を始まりとするインターバルにおいてAが実行されることを意味している（図10.a）．例えばAの値をインターバルの間Bに代入を行いたい場合には

KEEP（B：＝A）

と記述される．ただしインターバルの最後の時刻では代入は行われない（図10.b）．

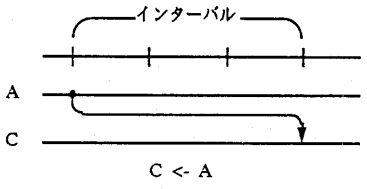


図6　時相代入文

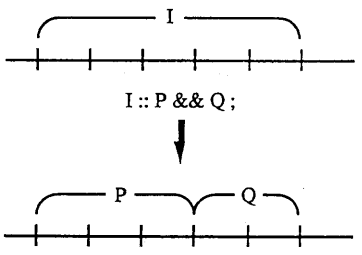
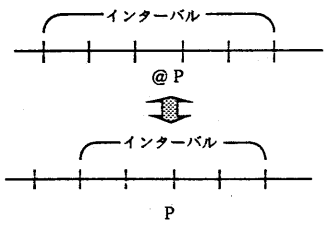
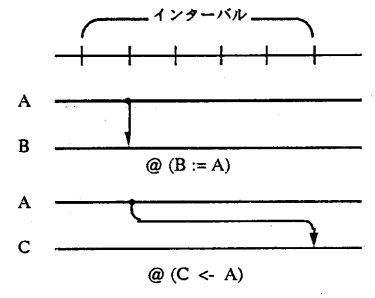


図7　&&によるインターバルの分割



(a) @の意味



(b) @を用いた代入

図8　NEXT演算子@

4.9 反復動作記述

繰り返し動作の記述はWHILE文，REPEAT文，FOR文によって行われるが，いずれもインターバルを最小単位として繰り返すものである．複数のインターバルを一つの端子として繰り返してもよい．

WHILE文とREPEAT文は指定された条件が成立するまでインターバルの動作を繰り返すものであるが，WHILE文では条件の判定はインターバルの始まりの時刻に実行するのに対し，REPEAT文ではインターバルの実行が終えた時点で行われる．そのためにWHILE文では最初に条件が成立すると一度もインターバルを実行せずに抜けてしまうことになる．WHILE文とREPEAT文の動作の違いについては図11に示す．

FOR文では指定した回数だけ動作を繰り返す．

```
FOR　変数＝初期値　TO　終値　BY　差分
DO
    インターバル記述
END-DO；
```

変数の値はインターバルが繰り返される度に差分で指定された分だけ増加し，変数の値が終値以上になるまで繰り返される．

4.10 その他

動作記述の基本構成を成しているFUNCTIONは起動されてはじめて実行され，FUNCTIONが終了すると動作も終了してまう．これに対して現モジュールにおいて常に実行されているような動作はBOOL記述で記述されるされる．BOOL記述はFUNCTIONの外に記述され次のような形式をとる．

BOOLEAN：実行文，・・・；

ここで実行文として記述できるのはIF文やCASE文といった条件式や現在代入文のみである．例えば組み合せ回路はここに記述される．

条件文としてはIF文やCASE文のほか真理値表文が記述できる．真理値表文を用いることである入力パターンに対する出力を真理値表の形で簡潔に記述することができる．またCASE文では入力パターンに対応したパターンが見つかるとそれ以上検索はせずに実行文を実行してCASE文を終了する．

またインターバルが終了したあと次に起動するインターバルはGOTO文を用いて指定される．

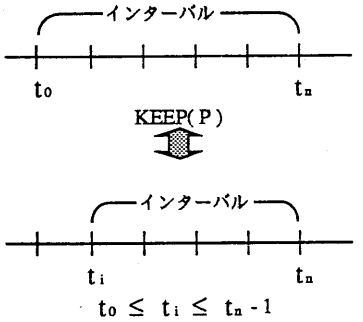
GOTO　インターバル名

ただしGOTO文が省略された場合は次に記述されたインターバルが自動的に起動される．

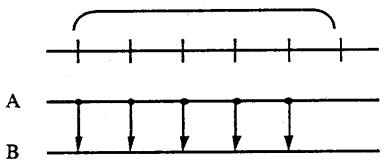
5. データパスview

データパスviewでは現モジュールで使用されるレジスタやバスなどのファシリティの宣言やデータパスの指定を行う．

ファシリティ宣言では各ファシリティのタイプおよびその名前が指定される．タイプにはREGISTER，STACK，BUSなどのようにシステム側で定義しているタイプとユーザーが定義するタイプの二種類が存在する．ユーザー定義のタ

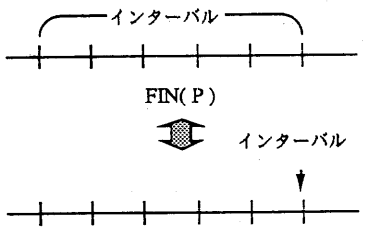


(a) KEEPの意味

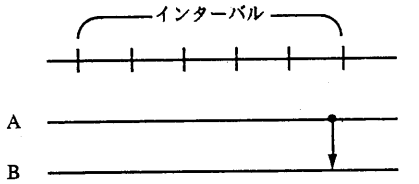


(b) KEEPをもちいた代入

図9 FIN演算子



(a) FINの意味



(b) FINをもちいた代入

図10 KEEP演算子

イプについては，その機能はUHDLで記述されタイプ名としてはそのモジュール名が指定される．

ファシリティ間のデータの流れを表すデータパスは，各ネット毎にネットへの入力およびネットからの出力につながっているファシリティの端子を指定して表現される．

6. その他のview

現モジュールの構造を指定したいような場合は，構造viewで指定される．例えばあるモジュールが図12のように複数のモジュールから構成されている場合を考えよう．この場合構造viewで記述すると図13のようになる．タイプ文は次のような構成をとっており，view構造で用いられるモジュールが指定される．

TYPES；

モジュールタイプ：固有名，・・・；

.

.

END-TYPES；

モジュールタイプはモジュールの名前であり，固有名は構造view内でのモジュールの名前である．続くネット記述において各モジュールの外部端子間の論理結線情報が記述される．ネット記述は次のような形式をとる．

ネット名：出力端子，入力端子，・・・；

出力端子はネットへ信号を送る端子であり，出力端子はネットから信号を受け取る端子である．

またUHDL記述全体を管理するviewとして管理viewがある．管理viewではシステム名，版，日付，設計者が記述される．

7. 記述例（Serial - Parallel Data Converter）

シリアルーパラレル(S－P)データコンバータの記述例を図14に示す．ここで扱うS－Pデータコンバータはシリアルデータを９ビットのパラレルデータに変換して出力する

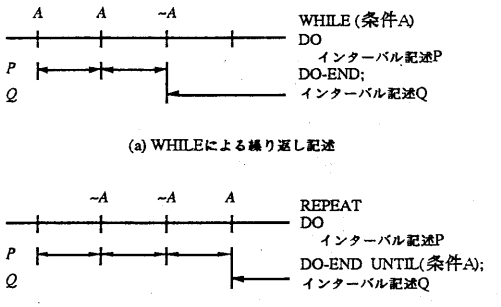


図11 WHILE文とREPEAT文

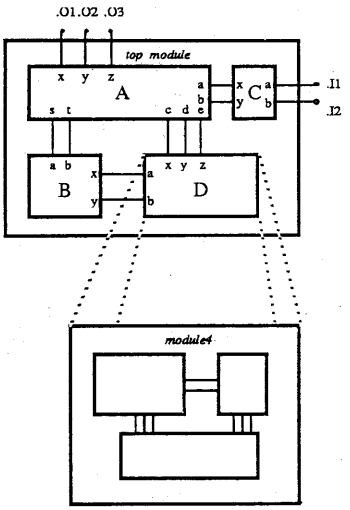


図12 階層構造を持つ回路

```
STRUCTURE-VIEW: top_module;
        PURPOSE: EXAMPLE;
        REVISION: 007;
        DATE: 88/11/22;
        DESIGNER: James;

        INPUTS: .I1, .I2;
        OUTPUTS: .O1, .O2, .O3;

        TYPES;
                module1 : A;
                module2 : B;
                module3 : C;
                module4 : D;
        END-TYPES;

        NETS;
                N01 :  .I1,   C.a;
                N02 :  .I2,   C.b;
                N03 :  C.x,   A.a;
                N04 :  C.y,   A.b;
                N05 :  D.x,   A.c;
                N06 :  D.y,   A.d;
                N07 :  D.z,   A.e;
                N08 :  B.x,   D.a;
                N09 :  B.y,   D.b;
                N10 :  A.s,   B.a;
                N11 :  A.t,   B.b;
                N12 :  A.x,   .O1;
                N13 :  A.y,   .O2;
                N14 :  A.z,   .O3;
        END-NETS;
END-VIEW;
```

図13 構造viewの記述例

ものである．シリアルデータの境界を認識するためにデータの始まりを表すSYNCBYTEのビットパターンをチェックした後でデータの変換を行っている．また変換されたデータはいったんバッファに蓄えられ，外部信号READITが反転し外部の信号受入体制が整ったことがわかると，バッファの値を出力する．このコンバータの動作を図15［4］に示す．

8. まとめ

ハードウェア記述言語UHDLについて説明してきた．現在UHDLを柱とした論理回路自動合成システムについて研究を行っており，目下シミュレータとUHDLを入力としてステートマシンに変換するトランスレータの開発に取り組んでいる．

参考文献

[1] B. Moszkowski, "Reasoning about Digital Circuits", STAN-CS-83-970, Dept. of Computer Science, Stanford University, June, 1983.

[2] B. Moszkowski, "A Temporal Logic for Multilevel Reasoning about Hardware", IEEE Computer Magazine, February,1985.

[3] J. R. Juley et al. "A Digital System Design Language (DDL)", IEEE Trans. on Computer, Vol.C-17, No.9, 1968.

[4] Franklin. P. Prosser, The Art of Digital Design, Englewood Cliffs, NJ: Prentice-Hall, 1987.

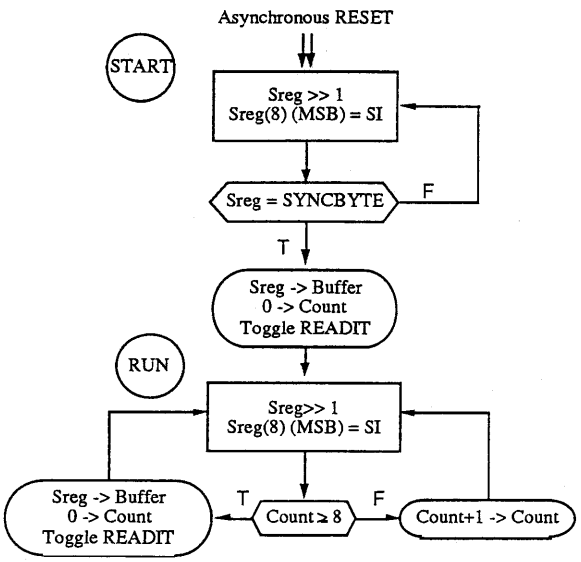


図14　S-P データ・コンバータのASMチャート

```
UHDL;

MANAGE-VIEW: AMAEBI;
SYSTEM: SYS1;
REVISION: 01;
DATE: 88/02/02;
DESIGNER: Drunken_Piper;
END-VIEW;

NAME: S2P-CONVERTER;

INTERFACE-VIEW: KOHADA;
PURPOSE: EXAMPLE;
REVISION: 01;
DATE : 88/02/02;
DESIGNER : Drunken_Piper;
INPUTS: .RESET, .SI;
OUTPUTS: .PO(9), .READIT;
END-VIEW;

DATAPATH-VIEW : AOYAGI;
PURPOSE: EXAMPLE;
REVISION: 01;
DATE : 88/02/02;
DESIGNER : Drunken_Piper;
TYPES;
    REGISTER(9): Sreg, Buf;
    COMP(9): SyncComp, CntComp;
    COUNTER(4): Count;
END-TYPES;
NETS;
    N0 = FROM(.SI)  TO(Sreg.IN8);
    N1 = FROM(Sreg.OUT)  TO(Buf.IN, SyncComp.IN);
    N2 = FROM(Buf.OUT)  TO(.SPBYTE);
    N3 = FROM(Count.OUT)  TO(CntComp.IN);
END-NETS;
END-VIEW;

BEHAVIOR-VIEW : AKAGAI;
PURPOSE: EXAMPLE;
REVISION: 01;
DATE: 88/02/02;
DESIGNER: Drunken_Piper;
DEFINE: SYNCBYTE = X'A5';              % for example
BOOLEAN: .READIT = myReadIt, .PO = Buf;
CLOCK: SPCLK(100000, 50000);
FUNCTION : MAIN : SPCLK;
LOGIC:: IF .RESET THEN (GOTO START);
START:
    Sreg <- (.SI || Sreg) >> 1,
    IF (Sreg = SYNCBYTE)
    THEN(       Buf <- Sreg,
        Count <- 0,
        myReadIt <-~myReadIt,
        GOTO RUN);
RUN:
    Sreg <- (.SI || Sreg) >> 1,
    IF (Count >= 8)
    THEN (Count <- 0,
        myReadIt <- ~myReadIt,
        Buf <- Sreg,
        GOTO RUN)
    ELSE (Count <- Count + 1, GOTO RUN);
FEND;

END-VIEW;
END-NAME;
END-UHDL;
```

図15　S-P・データ・コンバータのUHDL記述例